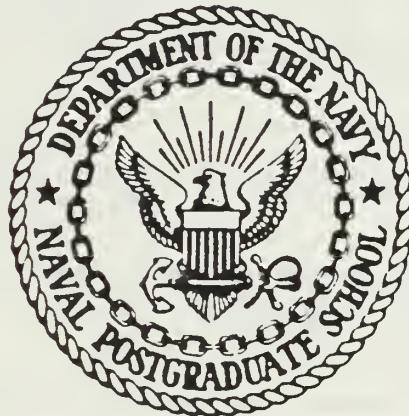


WAVES P 001
MONTEREY, CALIFORNIA 689-5302

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

1144

COMPUTER AIDED DESIGN FOR LINEAR CONTROL
STATE VARIABLE SYSTEMS (SVS)

by

Ismail Unlu

• • •

December 1987

Thesis Advisor:

George J. Thaler

Approved for Public Release; Distribution Unlimited

T239301

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b RESTRICTIVE MARKINGS	
2 SECURITY CLASSIFICATION AUTHORITY		3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution Unlimited	
4 DECLASSIFICATION/DOWNGRADING SCHEDULE			
5a PERFORMING ORGANIZATION REPORT NUMBER(S)		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable) Code 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
8a ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	
9a NAME OF FUNDING / SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
10a ADDRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS	
		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) COMPUTER AIDED DESIGN FOR LINEAR CONTROL STATE VARIABLE SYSTEMS (SVS)			
12 PERSONAL AUTHOR(S) Inlu, Ismail			
13a TYPE OF REPORT Master's Thesis	13b TIME COVERED FROM TO	14. DATE OF REPORT (Year, Month, Day) 1987 December	15 PAGE COUNT 164
16 SUPPLEMENTARY NOTATION			
COSATI CODES		18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	
19 ABSTRACT (Continue on reverse if necessary and identify by block number) The theory, detailed outline of operating and algorithm procedures of a continuous time-invariant, linear control state variable systems design and analysis computer program is presented. The program, SVS, which is based on Melsa's LINCON, was modified to demonstrate Controllability, Observability, Bode Plot, Root locus plot, Nyquist plot, pole placement, Luenberger observer design, optimal control design, time response plot and some basic matrix manipulations. Worked examples with the program output are included. Some options give only numeric data output; others give both numeric data and high-resolution graphs. The software, which is fully interactive, menu driven and user friendly is written in Turbo Pascal to be run on the IBM-PC microcomputers. All options are presented via option menus and the user will be prompted for all input parameters.			
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS		21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL Professor George J. Thaler		22b TELEPHONE (Include Area Code) (408) 646-2134	22c OFFICE SYMBOL Code 62TR

Approved for public release; distribution is unlimited.

Computer Aided Design for Linear Control
State Variable Systems (SVS)

by

Ismail Unlu
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

December 1987



ABSTRACT

The theory, detailed outline of operating and algorithm procedures of a continuous time-invariant, linear control state variable systems design and analysis computer program is presented. The program, SVS, which is based on Melsa's LINCON, was modified to demonstrate Controllability, Observability, Bode Plot, Root locus plot, Nyquist plot, pole placement, Luenberger observer design, optimal control design, time response plot and some basic matrix manipulations. Worked examples with the program output are included. Some options give only numeric data output; others give both numeric data and high-resolution graphs. The software, which is fully interactive, menu driven and user friendly is written in Turbo Pascal to be run on the IBM-PC microcomputers. All options are presented via option menus and the user will be prompted for all input parameters.

7-15-5
1000
101

TABLE OF CONTENTS

I.	LINEAR CONTROL STATE VARIABLE SYSTEMS (SVS) .	8
A.	INTRODUCTION TO THE SVS	8
B.	SYSTEM REQUIREMENTS	8
C.	FILES ON THE DISK	9
D.	STARTING TO THE SVS	10
II.	INPUT/CHANGE MENU	12
A.	INTRODUCTION TO THE INPUT/CHANGE MENU .	12
B.	INPUT PLANT MATRICES	14
C.	CHANGE CURRENT PLANT MATRICES	15
D.	SAVE CURRENT MATRICES TO DISK FILE . . .	16
E.	LOAD PLANT MATRICES FROM DISK FILE . . .	16
III.	GRAPHICS MENU	18
A.	INTRODUCTION TO THE GRAPHICS MENU . . .	18
B.	BODE PLOT	18
C.	TIME RESPONSE PLOT	23
D.	NYQUIST PLOT	24
E.	ROOT LOCUS PLOT	27
F.	CHARACTERISTIC EQUATION ROOTS	28
G.	LOAD PLANT MATRICES FROM DISK FILE . .	28
IV.	SVS MAIN MENU	30
A.	INTRODUCTION TO THE SVS MAIN MENU . . .	30
B.	CONTROLLABILITY	30
C.	OBSERVABILITY	32
D.	POLE PLACEMENT	33
E.	LUENBERGER OBSERVER DESIGN	40
F.	DESIGN OF OPTIMAL CONTROL	46
G.	MATRIX MATHEMATICS MENU	49

V.	CONCLUSION AND RECOMMENDATIONS.	52
APPENDIX A	UTILITI FILES	53
APPENDIX B	PROGRAM LISTING	54
LIST OF REFERENCES		158
INITIAL DISTRIBUTION LIST		161

LIST OF FIGURES

2.1	The SVS main menu	12
2.2	The input/change menu	13
2.3	Change current plant matrices selection . . .	15
2.4	Change current plant matrices example . . .	17
3.1	The graphics menu	18
3.2	Bode plot parameters selection	19
3.3	Open loop Bode plot for example 3.1 . . .	22
3.4	Closed loop Bode plot for example 3.1 . . .	23
3.5	Time response parameters screen	24
3.6	Time response plot for example 3.2	25
3.7	Parameters selection for the Nyquist plot	26
3.8	Example of the Nyquist plot	26
3.9	Parameters screen for the root locus	27
3.10	Example for the root locus plot	28
3.11	Example of characteristic equation roots . .	29
4.1	Controllability program output	32
4.2	Observability program output	33
4.3	Block diagram for the state variable representation	34
4.4	General closed loop system with state variable feedback	34
4.5	Linear state variable feedback system	35
4.6	Closed loop block diagram representation . .	36
4.7	Program output for the pole placement	38
4.8	Luenberger observer block diagram	40
4.9	Luenberger observer design parameters	41
4.10	Pole placement result for the Luenberger observer design	43
4.11	Observability result for Luenberger	44
4.12	Controllability output for Luenberger	44
4.13	The Luenberger observer design output	45
4.14	Optimal control parameters screen	48

4.15	Optimal control graphic output #1	49
4.16	Optimal control graphic output #2	50
4.17	Optimal control numerical output	51

I. LINEAR CONTROL STATE VARIABLE SYSTEMS

(SVS)

A. INTRODUCTION TO THE SVS

SVS is software for the IBM-PC microcomputer in the analysis and design of continuous time, linear control systems. These programs are based on the matrix mathematics of state variables and were first developed by Melsa [Ref. 1] and adapted for batch use at NPS by Desjardins [Ref. 2].

The original intent of this thesis was simply to take Desjardins' adapted version of Melsa's LINCON and modify it for the Turbo Pascal computer language. The features of this thesis are:

- User-friendly as possible
- Menu-driven program. <Q>quit key always returns to the SVS main menu.

The hierarchical menu structure is three levels deep at any point. So the user, before selecting his/her option, can get help at every menu.

SVS was tested with several examples and is now available to any user on the Naval PostGraduate School control laboratory PC's under the SVS directory name.

B. SYSTEM REQUIREMENTS

SVS is a large program and requires at least 512KB of Memory to run. The program will run on any IBM-PC or compatible "MS-DOS" computer and requires a standard IBM color Graphics Adapter (CGA) or IBM Enhanced Graphics Adapter (EGA) card. It will run on either monochrome or color monitor, but all menus are supported by color, for that reason they are easier to work with a color monitor if available. The graphics in the program are in high resolution (640 X 200) mode. They only appear in the white-on-black. The graphics

can be dumped to an EPSON, IBM-Graphics, or compatible printer by using the Shift + PrtSc key.

The program is written in the Turbo Pascal Language. Turbo Pascal has a 64K data segment and code sizes limitation. Under this restriction, SVS is compiled as five executable programs and nineteen "Chain" files. The main program name is SVS in the disk files. It has extension .COM. The chain files have extension .CHN and they are not themselves executable.

C. FILES ON THE DISK

A complete list of the files and a brief description of these are below.

- (1) SVS.COM { The executable main menu module }
- (2) INPUT.COM { The input/change menu module }
- (3) MATRIX.COM { The matrix mathematics menu module }
- (4) PLOT.COM { The graphics menu module }
- (5) CHANGE.CHN { Allows user to change input data
{ values }
- (6) POLYNOM.CHN { Calculates characteristic
{ polynomial of A matrix }
- (7) CONTROL.CHN { Calculates controllability of the
{ system }
- (8) DETERMIN.CHN { Calculates determinant of the A
{ matrix }
- (9) EIGEN.CHN { Shows eigenvalues of the A matrix }
- (10) BODE.CHN { Gives Bode plot of the system }
- (11) TIMEPLOT.CHN { Gives time response of the system }
- (12) OBSER.CHN { Calculates observability of the
{ system }
- (13) LUENBERG.CHN { Design Luenberger observer to
{ achieve a closed-loop poles of the
{ system }
- (14) OPTIMAL.CHN { Optimal control design program }

(15)POLE.COM	{ To calculate feedback coefficients { to achieve a desired closed-loop { poles }
(16)SAVE.CHN	{ Saves data to desired drive }
(17)RETRIEVE.CHN	{ Gets data from desired drive }
(18)INVERSE.CHN	{ To calculate inverse of the A { matrix }
(19)INPUTDAT.CHN	{ To permit user to enter the input { data for whole system }
(20)HELP1.CHN	{ Help program for SVS main menu { options}
(21)HELP2.CHN	{ Help information for the { input/change menu options }
(22)HELP3.CHN	{ Help information for the matrix { mathematics menu option }
(23)HELP4.CHN	{ Help information for graphics menu { option }
(24)NYQUIST.CHN	{ to calculate Nyquist (polar) plot { of the system }
(25)RLOCI.CHN	{ Root locus plotting procedure}
(26)ROOTS.CHN	{ Shows plant characteristic { equation roots }

In addition to these files, there are three "system" files that are needed to run the program. These are 4X6.FON, 8X8.FON and ERROR.MSG files.

D. STARTING TO THE SVS

This software package has two diskettes. The first step is to get the SVS main menu on your screen. All you have to do is follow these steps:

1. Turn on the power (or, if it's already on, the user reboots the computer).

2. Wait for the operating system prompt. It will look like one of the following:

C: _

C> _

C\> _

C:\> _

C:\> _

<C\> _

(or some other letter).

The prompt may look somewhat different, depending on the computer and how it has been set up.

3. Type **MD SVS** and press <ENTER> key. This opens a new directory.
4. Type **CD\SVS** and press <ENTER> key. This enters the new directory.
5. Insert disk #1 into disk driver A:
Type **Copy A:*. *** and press <ENTER> key.
6. Repeat step 5 for disk #2.
7. Type **SVS** and press <ENTER> key.

Now the user is in the SVS main menu and ready to work. Make a choice for further step.

II. INPUT/CHANGE MENU

A. INTRODUCTION TO THE INPUT/CHANGE MENU

The first step, of course, is to enter the A, B and C matrices into the computer as a common input. This is the starting point of the program. For our case, the general state variable equations are represented by the following equations.

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A} \mathbf{x}(t) + \mathbf{B} \mathbf{u}(t) & (2-1) \\ \mathbf{y}(t) &= \mathbf{C} \mathbf{x}(t) + \mathbf{D} \mathbf{u}(t) & (2-2)\end{aligned}$$

where A, B, C are given matrices and D is assumed to have a zero elements for this program.

*** SVS MAIN MENU ***	
(I) Input/Change Plant Matrices Menu	
(G) Graphics Menu	
(C) Controllability	
(O) Observability	
(L) Luenberger Observer Design	
(D) Design of Optimal Control	
(P) Pole Placement	
(M) Matrix Mathematics Menu	
(H) Help	
(Q) Quit the Program	
Make Your Selection_	
Naval Postgraduate School	Ismail UNLU

Figure 2.1 The SVS main menu.

From the opening menu of SVS, shown in Figure 2.1, we choose the "Input/Change Plant Matrices Menu" option to bring us to the Input/Change menu shown in Figure 2.2.

*** INPUT / CHANGE MENU ***
(I) Input Plant Matrices (C) Change Current Plant Matrices (S) Save Plant Matrices to Disk File (L) Load Plant Matrices From Disk File (H) Help (Q) Quit to SVS Main Menu
Make Your Selection

Figure 2.2 The input/change menu.

This input/change menu allows the user to enter common inputs to all programs. These common inputs are the plant or A matrix, the input or B matrix and the output or C matrix. The only restriction is the dimensions of the matrix. It must be no greater than 10. This means that the maximum matrix size has to be 10X10. However, due to a user decision, a dimension size not to exceed 6 is required. The reasoning behind this was due, in part, to the printer. Since the output format to the printer is E11, this naturally limits us to 6 numbers per line. I2 format places are

normally considered necessary for good accuracy. For systems with order greater than 6, every row of the matrix is continued on the second line. After attempting this, it was decided the results were difficult to read. Otherwise it is appropriate up to a 10th order system.

B. INPUT PLANT MATRICES

This option is used to initially enter the A, B and C matrices of the state variable equations (2-1) and (2-2). For this option, select the "Input plant matrices" from the input/change menu. The screen will prompt the degree of the plant, which is the dimension of the A matrix. The maximum acceptable degree is 10. Then it asks for elements of the A matrix and so forth.

Matrices are entered one element at a time beginning with 1,1 and continuing across the row of the matrix. The next row is then entered, and the process continues until all elements have been entered. After the matrix is entered, the complete matrix is automatically brought to the screen for review and possible element changes. If a change to the matrix is desired the user simply enters the row and column number to change after every prompt. Then the program asks for corresponding matrix elements to change. After being prompted, the change is entered. A review of the matrix is again brought to the screen. The user is again prompted for any more possible changes. This procedure continues until all changes have been done. The same A matrix procedures are repeated for the B and C matrices. This input data can be used for the options without saving to disk file until quitting the program.

C. CHANGE CURRENT PLANT MATRICES

This option allows the user to change input data A, B and C matrices quickly and easily. The user can change the order and elements of the matrices that were previously entered. This powerful combination facilitates both input correction and changes, especially for higher order matrices during the design process. First Figure 2.3 appears on the screen. The user can choose one matrix at a time for correction. Then the program gives the order and elements of the selected matrix. In the beginning the user can enter a

```
*** Change Current Plant Matrices Procedure ***
=====

Which matrix do you want to change ?  PLANT (A) A_
                                         INPUT (B)
                                         OUTPUT (C)


Press <ESC> to change it!,
Then input your choice with <ENTER> key
```

Figure 2.3 Change current plant matrices selection.

correction to the order of the matrix. Then, under the new dimension, can make corrections on elements of the matrix. The program shows the corrected results on the screen. At the end of the program, the user automatically returns to the input/change menu. If the user wants to change more than one matrix, he/she must

choose the "Change Current Plant Matrices" option two or more times. A basic example for this option is illustrated in Figure 2.4.

D. SAVE CURRENT MATRICES TO DISK FILE

This procedure is used to store plant matrices to the hard disk or floppy disk file. First it prompts drive C as a saving drive. Then asks the user for the drive designator (A through E), and filename for the problem to be saved. Eight characters of a MS-DOS filename are allowed; the program gives a filename extension of ".SVS" to each data. This extension is used to limit the disk search for appropriate files. A drive and filename are supplied by the program, which opens the file and stores the data. The procedure stores the data as a text file. A text file consists of ASCII characters, and is usually designed to hold readable information [Ref. 3].

E. LOAD PLANT MATRICES FROM DISK FILE

The procedure first asks the data drive where the problems are stored. After this is done, the program calls another procedure called "Directory" [Ref. 4]. This procedure uses MS-DOS function, calls and shows all available data files on the screen. The user can choose one of the files by moving arrow keys with <RETURN> key. Then the program opens the file and reads it.

The directory displays only the disk files with the extension ".SVS". This eliminates the possibility to read other files.

The A matrix is :

0.0000E+00	1.0000E+00	0.0000E+00	0.0000E+00
0.0000E+00	0.0000E+00	1.0000E+00	0.0000E+00
0.0000E+00	0.0000E+00	0.0000E+00	1.0000E+00
0.0000E+00	-1.5000E+01	-2.3000E+01	-9.0000E+00

The order of the system is:4, Change ? (Y/N)

The order of the system is:3

The A Matrix is :

0.0000E+00	1.0000E+00	0.0000E+00
0.0000E+00	0.0000E+00	1.0000E+00
0.0000E+00	0.0000E+00	0.0000E+00

Do you want to change any element ? (Y/N)

Input row to change : 1

Input column to change : 1

A(1,1)= 10

The A Matrix is :

1.0000E+01	1.0000E+00	0.0000E+00
0.0000E+00	0.0000E+00	1.0000E+00
0.0000E+00	0.0000E+00	0.0000E+00

Do you want to change any element ? (Y/N)

Figure 2.4 Change current plant matrices example.

III. GRAPHICS MENU

A. INTRODUCTION TO THE GRAPHICS MENU

From the opening menu of SVS, shown in Figure 2.1, we choose the " Graphics Menu " option to bring us the Graphics menu shown in Figure 3.1. The Graphics menu contains five basic options. These are the time response plot, Nyquist plot, Root locus plot, characteristic equation roots and Bode plot. This menu is also supported with the "Load plant matrices from disk file" option. This selection allows the user to get data from the disk file quickly instead of going via the SVS main menu route.

*** GRAPHICS MENU ***
(L) Load Plant Matrices From disk File
(C) Characteristic Equation Roots
(B) Bode Plot
(N) Nyquist Plot
(T) Time Response Plot
(R) Root Locus Plot
 (H) Help
(Q) Quit to SVS Main Menu
 Make Your Selection

Figure 3.1 The graphics menu.

B. BODE PLOT

Bode plot analysis can be accomplished with the Graphics menu by selecting the Bode plot option. This selection brings Figure 3.2 to the screen. There

are two selections for the frequency plot: the open-loop Bode plot and closed-loop Bode plot. Input data for this option is entered with the Input/change plant matrices menu selection which is explained in Chapter II. If the user wants the closed-loop Bode plot, the program automatically calculates for the negative unity-feedback condition.

*** Bode Plotting Parameters ***	
=====	
Open (O) or Closed (C) Loop Plot?	0
What is the first frequency to be plotted? .1 (Example: .01, 1, 100, etc.)	
How many decades do you want plotted?	4

Figure 3.2 Bode plot parameters selection.

The user also must enter the starting frequency and number of decades for the plotting. The upper frequency limit is calculated based on the number of decades. That is, if user selects .1 as the starting frequency with 4 decades, then the upper frequency will be 10000 rad/sec.

Bode plot displays two plots at the same time. These are plots of magnitude and phase versus radian frequency. Magnitude is converted to the decibels unit using the relation

$$\text{Magnitude}_{dB} = 20 \log_{10}(\text{magnitude})$$

and phase is converted to degrees using the relation

$$\text{Phase}_{\text{degree}} = (180/\pi) \text{Phase}$$

Magnitude calculations for the single pole or zero can be written as

$$\text{Magnitude} = [\text{Realpart}^2 + (\omega - \text{Imaginarypart})^2]^{1/2}$$

and the phase calculation is

$$\text{Phase} = \text{Tan}^{-1}[(\omega - \text{Imaginarypart})/\text{Realpart}].$$

For the whole system, the magnitude and phase are calculated for each pole or zero. Then the final magnitude is

$$\text{Magnitude}_{\text{system}} = \text{Magnitude}_{\text{zeros}} / \text{Magnitude}_{\text{poles}}$$

and the phase is

$$\text{Phase}_{\text{system}} = \text{Phase}_{\text{zeros}} - \text{Phase}_{\text{poles}}$$

The plots of the magnitude and phase are shown on the same graph. Coordinate values of 0dB magnitude and -180° phase coincide in the graph. We know from control theory that phase margin is read at the zero crossover of the magnitude curve and the gain margin is read at the -180° crossover of the phase curve. These two values can be read directly from the graph. The Bode routine calculates the numbers required for the plots. The procedure "plot-Bode" converts the number to a graphical display. Within the Bode-plot routine is a call to the procedure "graph-menu". Graph-menu is called by all procedures which produce a graph. It provides a small menu offering the user the choice to add a title to the graph, print the graph on the

printer, print the number or quit and return to the menu.

If the user selects the title to graph and it is completed, the plot is displayed with the title on the screen. The title block can be moved by using the cursor arrow keys and relocated anywhere on the screen. When the title box is moved where the user wants, the <enter> key must be pressed. Then the screen is frozen in position and the graph menu is recalled on the screen.

The print numbers selection saves the current graphic screen and permits the user to print the numbers used to draw the graph. The numbers may be printed on the printer (this will use a lot of paper) or to a disk file. If the disk option is selected, the user can scan that file with a word processor or by using the DOS "type" command and examine the points of interest. This option is illustrated on example 3.1.

EXAMPLE 3.1

The example can be stated as follows: Given plant transfer function

$$G_p(s) = \frac{100 (0.02 s + 1)}{(s + 1) (.1 s + 1) (.01 s + 1)^2} \quad (3-1)$$

was rearranged as a state variable equations.

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1000000 & -1120000 & -12210 & -211 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 10000000 & 200000 & 0 & 0 \end{bmatrix} x(t)$$

- a) Obtain the Bode diagram of the above system.
- b) Mark the following on the Bode diagram, recording the numerical values.
 - 1) Gain crossover frequency
 - 2) Phase margin
 - 3) Phase crossover frequency
 - 4) Gain margin
 - 5) Resonant frequency.

Solution:

Given data entered to the program by the selecting of the "Input Plant Matrices" option in the input/Change menu. Then the program outputs can be seen from the Figure 3.3 and Figure 3.4.

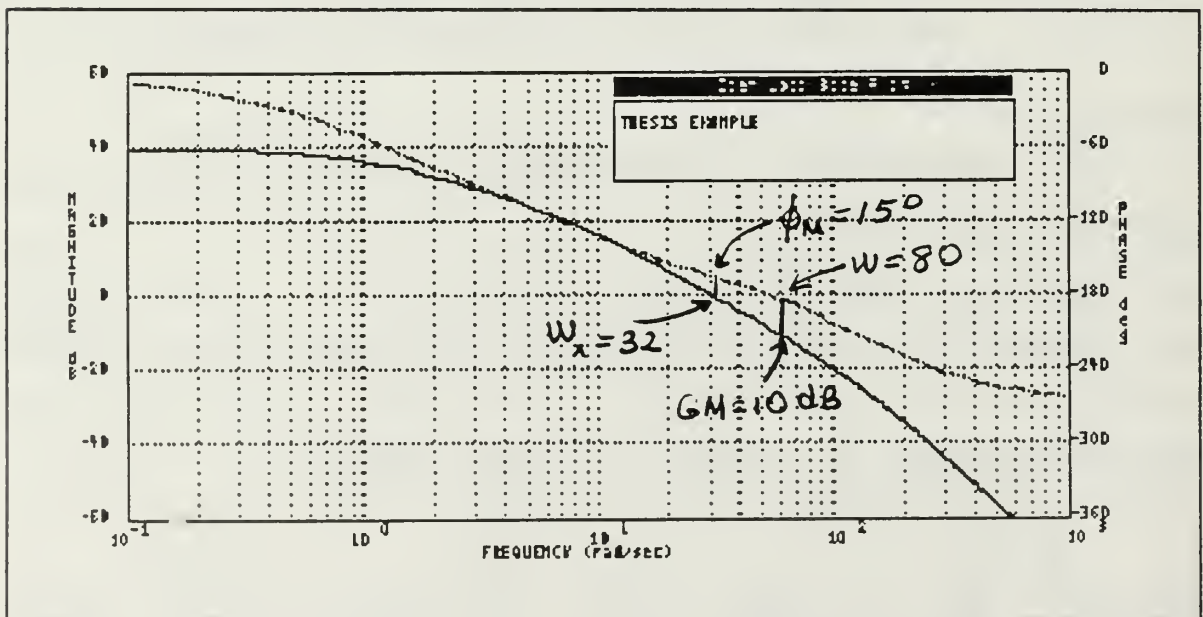


Figure 3.3 Open loop Bode plot for example 3.1.

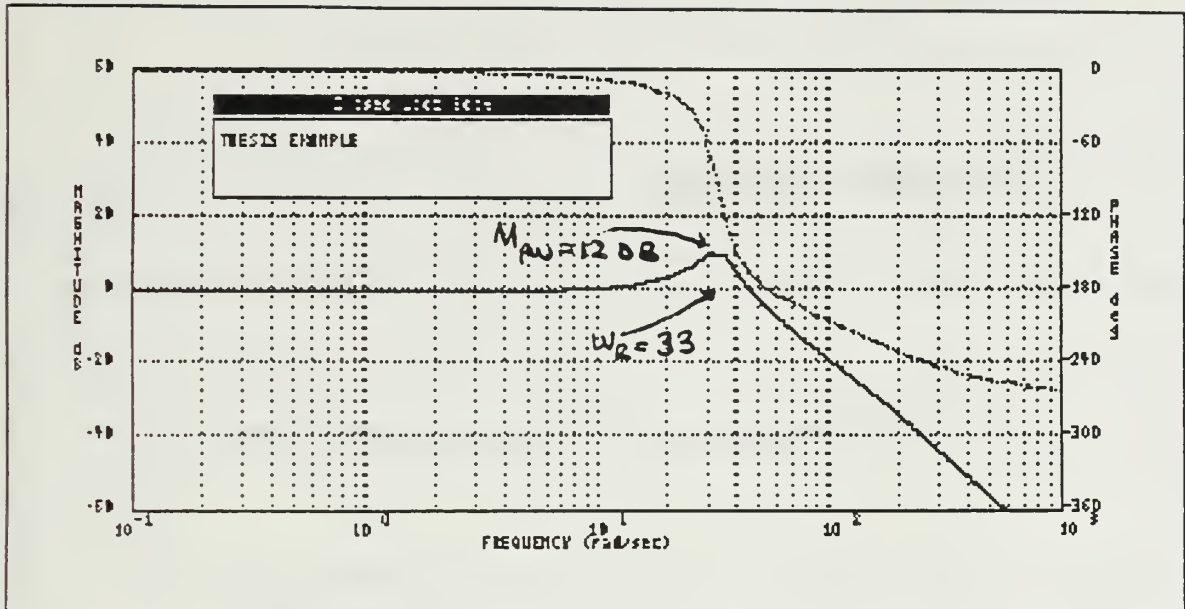


Figure 3.4 Closed loop Bode plot for example 3.1.

C. TIME RESPONSE PLOT

It is desirable to see the system's response in the time-domain to a typical input. Time response can be calculated and plotted with the SVS program as can the system input sinusoid, ramp, step or impulse. All these inputs have user selectable amplitudes. Figure 3.5 shows the time response parameter screen.

The time response algorithm first converts the A, B and C matrices to the open loop transfer function, then into a discrete-time, state-space equivalent. The theory of the time response plot is not included here. Users who want more information about the subject should consult reference 4, or any other relevant textbooks.

EXAMPLE 3.2

For the given system in example 3.1, obtain the time response plot and mark the following on the time response plot, recording the numerical values.

- 1) Settling time
- 2) Maximum overshoot for a step input.

Solution:

The program data has already entered for the Bode plot. The selecting "Time response Plot" in the graphics menu gives figure 3.6 as a problem solution.

```
*** Time Response Plotting Parameters ***
=====

What is your input to the system? STEP (S)  S
                                   RAMP (R)
                                   SINE WAVE (W)
                                   IMPULSE (I)

What is your input amplitude?      1

Input one of these choices, Open (O) or Closed (C) C

Input your simulation time to the system (99max) 1.4
```

Figure 3.5 Time response parameters screen.

D. NYQUIST PLOT

This section presents the Nyquist plot option. This selection gives open loop and unity-feedback closed loop Nyquist plot. The program first calculates open loop transfer function of the plant. Then the plot is obtained by calculating the magnitudes and phases angle of the transfer function for a specified

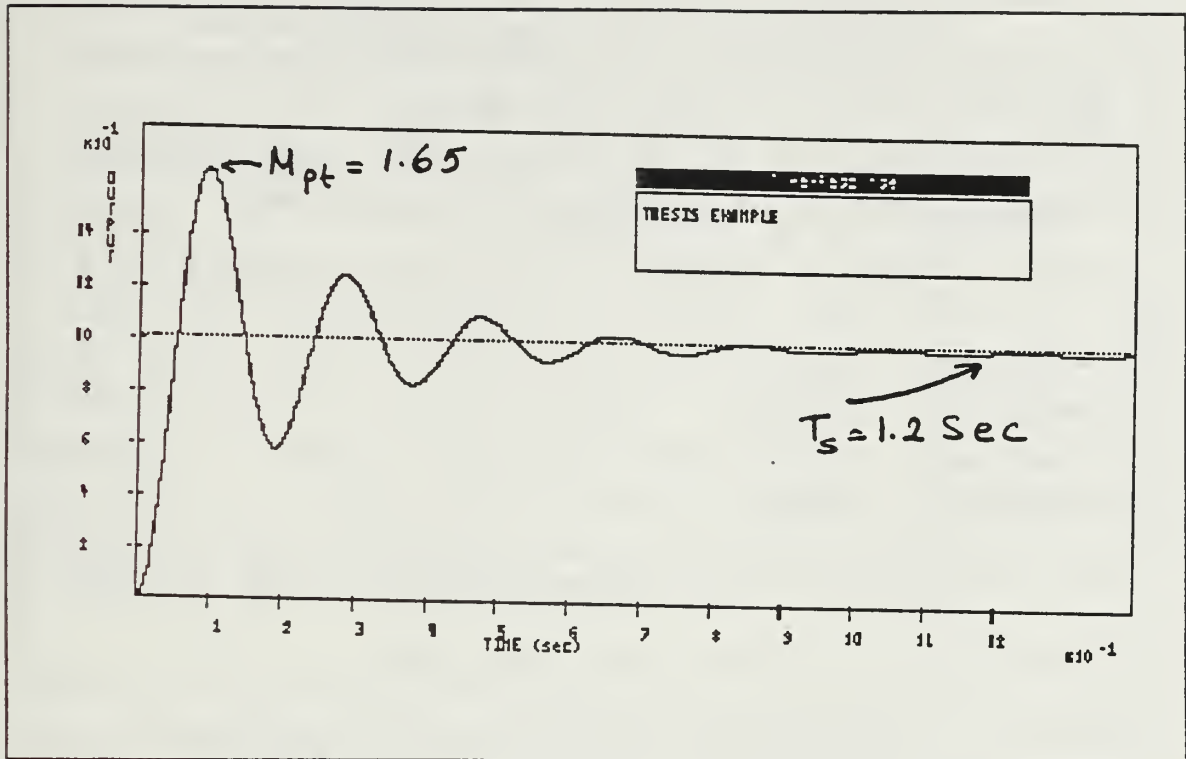


Figure 3.6 Time response plot for example 3.2.

number of times with a specified increment of w . Before proceeding to the Nyquist plot, the program prompts the menu of Figure 3.7. This parameters screen allows the user to enter additional data for the plotting. The graphic window size is given 100X100 scale if the user selects the big picture option. The select own size option asks the user to enter starting frequency, number of decades and X,Y coordinates maximum and minimum values for the plot. After getting the plot, the procedures are the same as with the Bode plot option. These are make a title to the graph, printer output and listing numbers (which are used to generate a graph) either to the printer or a specified file with a given drive name.

```

*** Nyquist Plotting Parameters ***

Open (O) or (C) Closed loop plot?    O
Graph window (B) or (S) Select your own size?  S
Input your first frequency to be plotted?      .1
(Example: .01, 1, 100, etc. )
Input number of decades do you want plotted?    4

X-Maximum      100
X-Minimum      -100
Y-Maximum      100
Y-Minimum      -100

Any changes to these parameters? ( Y / N )      N

Press <F1> to change previous entry

```

Figure 3.7 Parameters selection for the Nyquist plot.

EXAMPLE 3.3

Figure 3.8 shows the Nyquist plot for the example 3.1 A, B and C matrices.

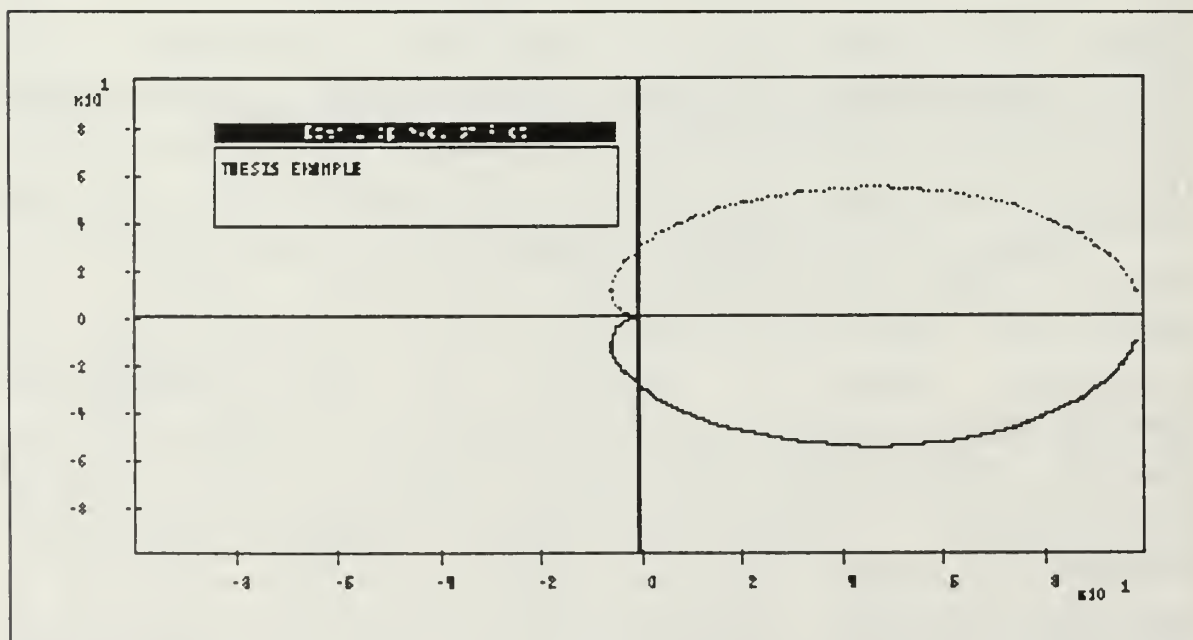


Figure 3.8 Example of the Nyquist plot.

E. ROOT LOCUS PLOT

This option plots the loci of the closed loop poles of a system with transfer function $G(s) = K N(s)/D(s)$ for varying gain. $N(s)$ and $D(s)$ are polynomials of the plant and the program calculates from the given A, B and C matrices.

The user is prompted to enter the starting and ending gain values, maximum and minimum X,Y coordinate values and to select either positive or negative feedback. These plotting parameters are shown on Figure 3.9. After this input routine, the program assumes unity feedback and calculates root locations for varying gain and plots them.

*** Root Locus Plotting Parameters ***	
Input STARTING value for the varying gain	0
Input ENDING value for the varying gain	10
X-Minimum	-100
X-Maximum	100
Y-Minimum	-100
Y-Maximum	100
Positive or Negative feedback? (P / N)	N
Any changes to these parameters?	N

Figure 3.9 Parameters screen for the root locus.

EXAMPLE 3.4

Figure 3.10 shows the Root locus plot for the given data on the example 3.1.

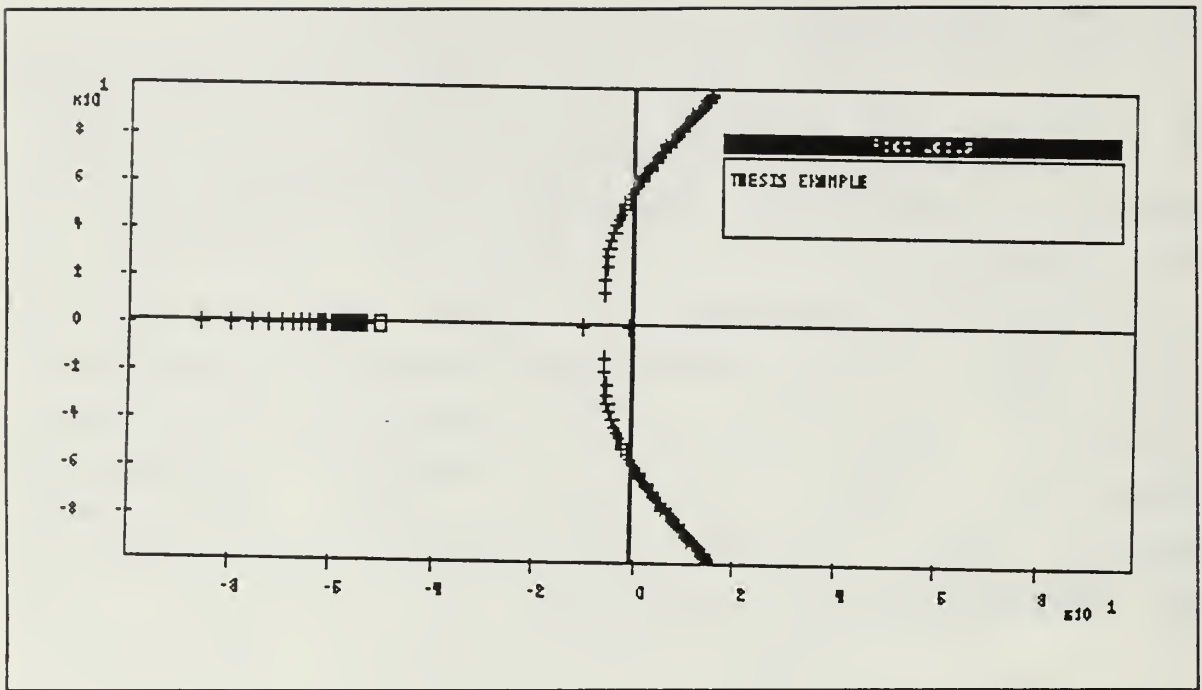


Figure 3.10 Example for the root locus plot.

F. CHARACTERISTIC EQUATION ROOTS

This option of the graphics menu allows the user to decide whether the system is stable or unstable by looking at the root location of the characteristic equation. The program gives again the unity-feedback closed loop characteristic equation roots of the plant. The illustrative program output of the example 3.1 can be seen in Figure 3.11.

G. LOAD PLANT MATRICES FROM DISK FILE

This last option of the graphics menu allows the user to get saved data from disk file directly instead of going via to the main menu and input/change menu.

```
*** Plant Characteristic Equation Roots ***  
  
ROOTS OF THE NUMERATOR  
  
s[1]= -50.00 +j 0.000  
  
ROOTS OF THE DENOMINATOR  
  
s[1]= -132.032 +j 0.000    s[2]= -69.170 +j 0.000  
s[3]= -4.899    +j -32.893 s[4]= -4.899 +j 32.899
```

Figure 3.11 Example of characteristic equation roots.

IV. SVS MAIN MENU

A. INTRODUCTION TO THE SVS MAIN MENU

In this chapter six options are presented which may be used for the analysis and design of control systems. These options are supported by the other options which are explained in Chapter II and III.

The observability option is used to determine the observability index of the system. Another very similar program, controllability is used to determine the controllability of the system. The matrix mathematics option brings to the screen another menu selections. This program calculates an A matrix determinant, inverse, characteristic polynomial and eigenvalues. The last three options may be used to design optimal linear control systems. The pole placement option is useful in the design of linear control state variable feedback control systems. In the pole placement case, the control is computed by multiplying by a gain $[K]$, the difference between the reference input and a weighted (linear) sum of the state variables. The Luenberger observer design is used to design a combined observer-controller to achieve a given desired closed loop transfer function when some of the states are not accessible.

Design of optimal control will minimize a given cost function which produces a scalar control. The program starts to work at the terminal time and works backwards in time.

B. CONTROLLABILITY

This option is used to determine the controllability of the linear time-invariant system. Consider the following continuous-time system

$$\dot{x}(t) = A x(t) + B u(t) \quad (4-1)$$

where

x = state vector
 A = plant matrix
 B = input matrix
 u = control input

The system described by the above equation is said to be state controllable at a given initial time if it is possible to construct an unconstrained control signal which will transfer an initial state to any final state in a finite time interval [Ref. 5].

This requires an algebraic condition such that the rank $r(C)$ of the controllability condition matrix

$$C = [B \mid AB \mid \dots \mid A^{n-1}B] \quad (4-2)$$

is n , the order of the system.

EXAMPLE 4.1

Consider the matrices A and B ,

$$A = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Determine if $[A, B]$ is a controllable pair.

Solution: Since A is 3×3 and B is 3×2 , matrix C has to be 3×6 . The program checked the rank of the controllability condition matrix. The result of the

program can be seen in Figure 4.1, the system is controllable.

CONTROLLABILITY		RESULT
The Plant matrix A is :		
0.000000E+00	1.000000E+00	-1.000000E+00
1.000000E+00	0.000000E+00	1.000000E+00
0.000000E+00	1.000000E+00	-1.000000E+00
The input Matrix B is :		
1.000000E+00	0.000000E+00	
0.000000E+00	1.000000E+00	
0.000000E+00	0.000000E+00	
The system is controllable.		

Figure 4.1 Controllability program output.

C. OBSERVABILITY

In this section we determine observability index of the linear systems. Consider the unforced system described by the following equations:

The (unforced) time invariant system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} \quad (4-3)$$

with the observation vector

$$\mathbf{y} = \mathbf{C}\mathbf{x} \quad (4-4)$$

The observability index, which is defined as the rank $r(0)$ of the observability condition matrix

$$0 = \begin{bmatrix} \mathbf{C} \\ \mathbf{C}\mathbf{A} \\ \vdots \\ \mathbf{C}\mathbf{A}^{n-1} \end{bmatrix} \quad (4-5)$$

is n , the order of the system.

The program output for this selection is illustrated in Figure 4.2 with the A matrix as given on example 4.1 and the following C matrix.

$$C = [1 \quad 0 \quad 0] \quad (4-6)$$

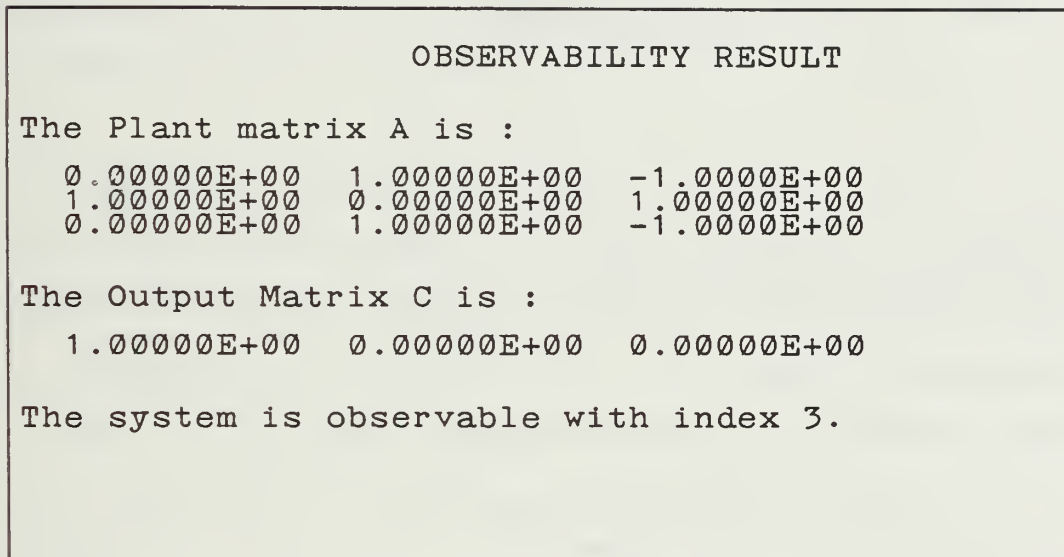


Figure 4.2 Observability program output.

D. POLE PLACEMENT

Any single-input single-output linear, time invariant system is described by the following equations:

$$\dot{x}(t) = A x(t) + B u(t) \quad (4-7)$$

$$y(t) = C x(t) \quad (4-8)$$

The plant is characterized as a block diagram in Figure 4.3.

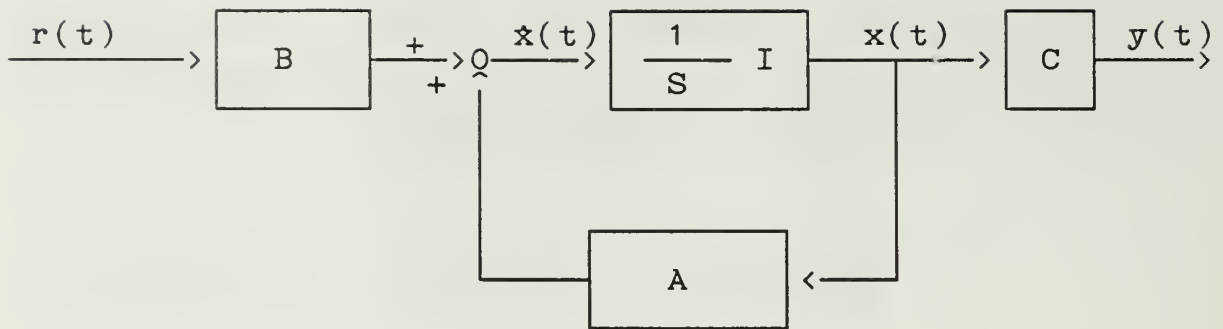


Figure 4.3 Block diagram for the state variable representation.

The closed loop nature of the system of Figure 4.4 is showed by the presence of the controller. It generates the control signal u from the knowledge of the state variables. So, we can see that, except for the reference input $[r]$, the state of the plant x is the only information needed by the controller. The

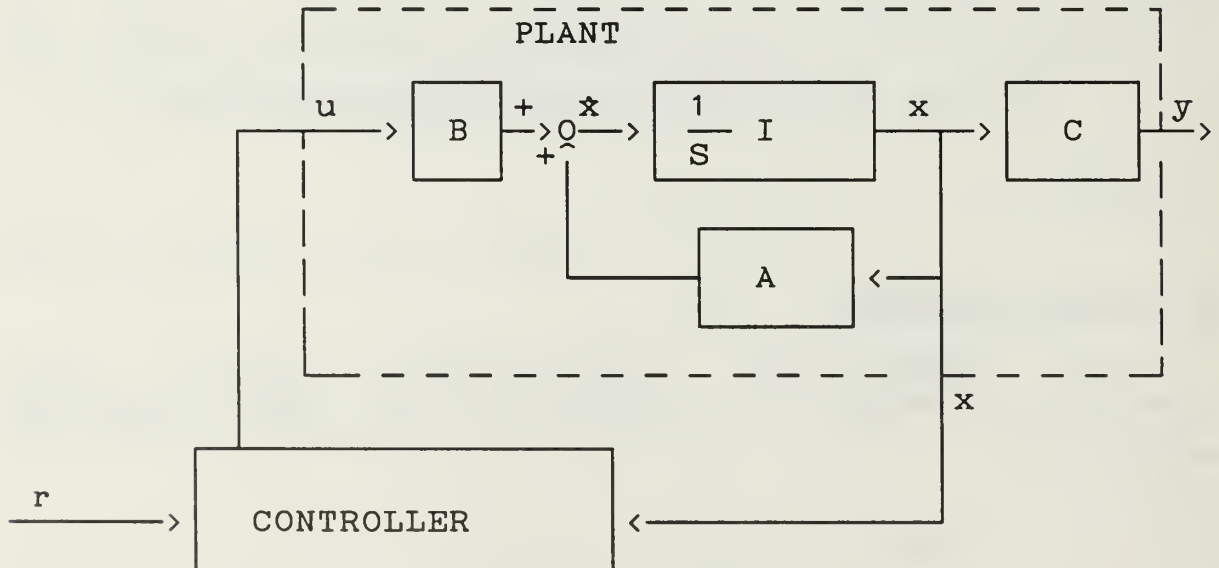


Figure 4.4 General closed loop system with state variable feedback.

control input is computed by the multiplying by a gain K the difference between the reference input and a weighted (linear) sum of the state variables. As a mathematical expression,

$$u = K [r - (k_1 x_1 + k_2 x_2 + \dots + k_n x_n)] \quad (4-9)$$

where the k_i 's are referred to as feedback coefficients. The gain K is referred to as the controller gain. The equation (4-9) may be simplified by making it in the matrix notation

$$u = K [r - k^T x] \quad (4-10)$$

After all this notation, the graphical representation of the system configuration is shown in Figure 4.5.

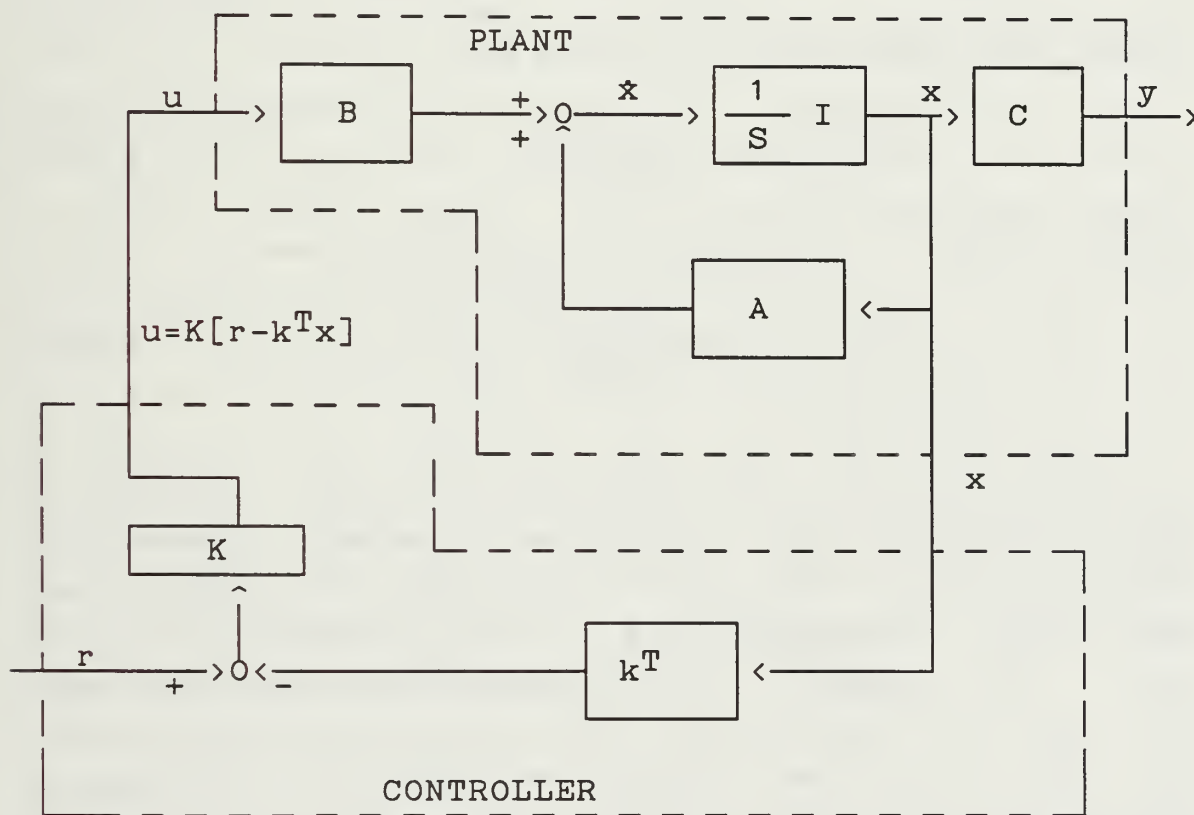


Figure 4.5 Linear state variable feedback system.

We could obtain the closed loop transfer function $Y(s)/R(s)$ from the state variables. Our approach is to force the system into the configuration shown in Figure 4.6.

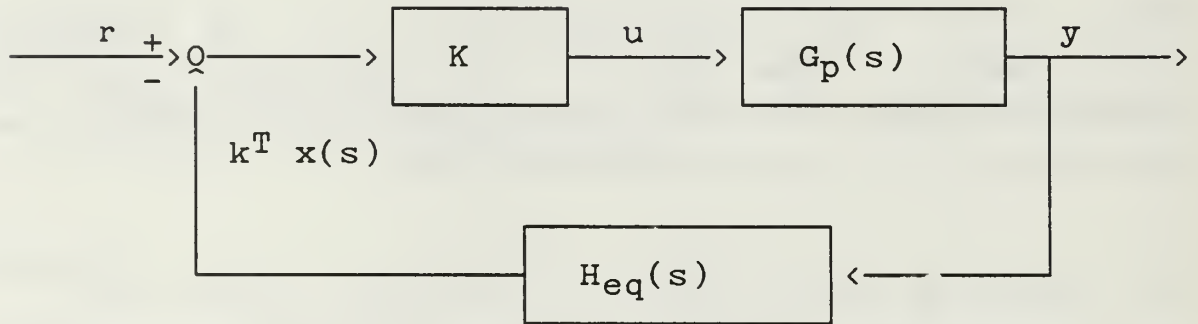


Figure 4.6 Closed loop block diagram representation.

We can see that $H_{eq}(s)$ is given by

$$H_{eq}(s) = \frac{k^T x(s)}{y(s)} \quad (4-11)$$

Since $y(s) = C x(s)$, this expression becomes

$$H_{eq}(s) = \frac{k^T x(s)}{C x(s)} \quad (4-12)$$

Here it must be pointed out that the program calculates the gain K , so zero steady state error results from a step input. If the user wants other conditions, he may rescale K and k^T appropriately by hand. For example, assume he wants to desire to have the controller gain, $K=K_1$, but the program output shows that $K=K_0$ with the feedback coefficients k_1, k_2, k_3 . The procedure is then to modify the program outputs by setting $K=K_1$ and setting

$$k^T = \frac{K_0}{K_1} [k_1 \ k_2 \ k_3] \quad (4-13)$$

This procedure does not change $Y(s)/R(s)$ and satisfies the condition $K=K_1$. Under these notations and block diagrams, this option gives an open loop transfer function $Y(s)/U(s)$ for the plant, the feedback transfer function $Heq(s)$, the controller gain $[K]$ and the feedback coefficients $[k^T]$ to achieve the desired closed loop characteristic polynomial.

The desired closed loop characteristic polynomial is the denominator of $Y(s)/R(s)$ and must agree with the order of the plant. The user can enter this polynomial either in coefficient form or factored form. If the user wants to enter the coefficient form, the coefficient of the highest degree term must be unity.

The methodology for computing the result is: the coefficients of the denominator polynomial of $Y(s)/R(s)$, which is the polynomial desired by the user, may be adjusted at will by proper selection of k and K . The closed loop zeros are equal to the open loop zeros. In other words, linear state variable feedback has no effect on the zeros of $Y(s)/R(s)$. The program calculates the numerator of $Heq(s)$ to achieve the desired characteristic polynomial. Note that the complete $Heq(s)$ is calculated by taking the numerator of $Heq(s)$ and dividing it by the numerator of $Y(s)/U(s)$. In the program procedures, the coefficients of the characteristic polynomial are computed by the use of the Principle-Minor method. For the matrix inverse calculation, the program uses the diagonalization procedures.

EXAMPLE 4.1

The plant matrices of a third order system are given in the following state variable representation:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} -1.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & -3.0 & 0.0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(t) \quad (4-14)$$

$$y(t) = [1.0 \quad 1.0 \quad 0.0] \mathbf{x}(t) \quad (4-15)$$

Find the feedforward (controller) gain $[K]$ and the feedback coefficients $[k^T]$ required to achieve a closed loop transfer function of

$$\frac{Y(s)}{R(s)} = \frac{2(s+2)}{s^3 + 4s^2 + 6s + 4} \quad (4-16)$$

The program result for this example is presented in Figure 4.7.

```

                                POLE PLACEMENT RESULT
The Plant matrix A is :
-1.00000E+00  1.00000E+00  0.00000E+00
 0.00000E+00  0.00000E+00  1.00000E+00
 0.00000E+00 -3.00000E+00  0.00000E+00
The Input matrix B is :
 0.00000E+00
 0.00000E+00
 1.00000E+00
The Output Matrix C is :
 1.00000E+00  1.00000E+00  0.00000E+00
Denominator of Y(s)/U(s) - Descending powers of S :
 1.0000  1.0000  3.0000  3.0000

```

Figure 4.7 Program output for the pole placement.

The poles of the $Y(s)/U(s)$ are:

REAL PART		IMAGINARY PART
-1.0000	+j	0.0000
0.0000	+j	-1.7321
0.0000	+j	1.7321

Numerator of $Y(s)/U(s)$ - Descending powers of S :

1.0000 2.0000

The zeros of the $Y(s)/U(s)$ are:

REAL PART		IMAGINARY PART
-2.0000	+j	0.0000

Desired closed-loop Characteristic polynomial -

Descending powers of S :

1.0000 4.0000 6.0000 4.0000

The roots of desired closed-loop characteristic polynomial are:

REAL PART		IMAGINARY PART
-2.0000	+j	0.0000
-1.0000	+j	-1.0000
-1.0000	+j	1.0000

Numerator of the $Heq(s)$ is - Descending powers of S :

1.5000 1.5000 0.5000

The roots of the $Heq(s)$ are :

REAL PART		IMAGINARY PART
-0.5000	+j	-0.2887
-0.5000	+j	0.2887

The feedback coefficients $[k^T]$ are :

0.5000 0.0000 1.5000

The gain $[K]$ is : 2.0000

Figure 4.7 Program output for the pole placement
(continued)

The results shown in Figure 4.7 specify that the gain of the controller $[K]$ is 2.0 and the feedback coefficients are $k_1=0.5$, $k_2=0.0$ and $k_3=1.5$.

E. LUENBERGER OBSERVER DESIGN

Consider a linear time invariant plant of the form

$$\dot{x}(t) = A x(t) + B u(t) \quad (4-17)$$

$$y(t) = C x(t) \quad (4-18)$$

Let a feedback control law for equations (4-17) and (4-18) will be

$$u(t) = K [r(t) - k^T x(t)] \quad (4-19)$$

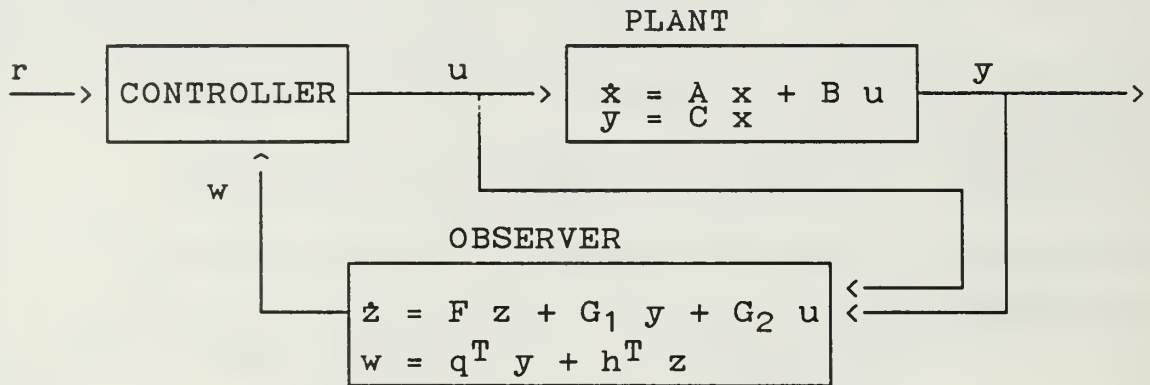


Figure 4.8 Luenberger observer block diagram.

Assume without loss of generality that the plant is controllable and observable. Since the state of equation (4-17) is not directly available to implement equation (4-19), an observer of the form

$$\dot{z}(t) = F z(t) + G_1 y(t) + G_2 u(t) \quad (4-20)$$

and replacing the true states with estimates yields

$$u(t) = K [r(t) - k^T z(t)] \quad (4-21)$$

$$k^T z(t) = h^T z(t) + q^T y(t) \quad (4-22)$$

will be designed. The resulting closed loop system is shown in Figure 4.8.

Where

$\mathbf{x}$ = state vector
 u = control input
 y = output
 r = system forcing input
 $\dot{\mathbf{z}}$ = estimated state vector
 A = plant matrix
 B = input matrix
 F = observer eigenvalues matrix
 G_1 and G_2 = observer gain matrices
 K = controller gain
 q^T = output feedback coefficients matrix
 h^T = observer feedback coefficients matrix

```
*** Luenberger Observer Design Parameters ***
=====
Input degree of observer (10 max)  2

Input the desired feedback coefficients in
                                     Factored <F> Form  C
                                     or Coefficient <C> Form

Input observer characteristic polynomial in
                                     Factored <F> Form  C
                                     or Coefficient <C> Form

Press <ESC> to change it!,
Then type your input with <ENTER> key
```

Figure 4.9 Luenberger observer design parameters.

The user has to enter the controller gain and the feedback coefficients which can be found by the use of

"pole placement" option. The program asks the observer eigenvalues which are represented by the F matrix in the program. The observer degree depends on the observability index. For example, if the observability index r is the minimum integer, then the observer gain matrix $[G]$ has order r . Simply the order of the observer, when the program prompted can be entered, As equal to or greater than $(r-1)$. These input parameters are shown in Figure 4.9.

EXAMPLE 4.2

The example presented here for the fourth degree plant is taken from Desjardin's [Ref. 2]. The plant is represented by the following equations.

$$y(t) = [10 \quad 20 \quad 0 \quad 0] x(t) \quad (4-23)$$

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -15 & -23 & -9 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u(t) \quad (4-24)$$

Solution:

Step1

The x_1 and x_2 are the only measurable states and we want to achieve following closed loop transfer function.

$$\frac{Y(s)}{R(s)} = \frac{1}{s^4 + 6s^3 + 17s^2 + 28s + 20} \quad (4-25)$$

The controller gain $[K]$ and the feedback coefficients required are found, as can be seen from Figure 4.10 by the use of the "Pole placement" option in the same menu. Results shown in Figure 4.10 indicates that the feedback coefficients $[k^T]$ are -3, -6, 13, 20 and the controller gain $[K]$ equals unity.

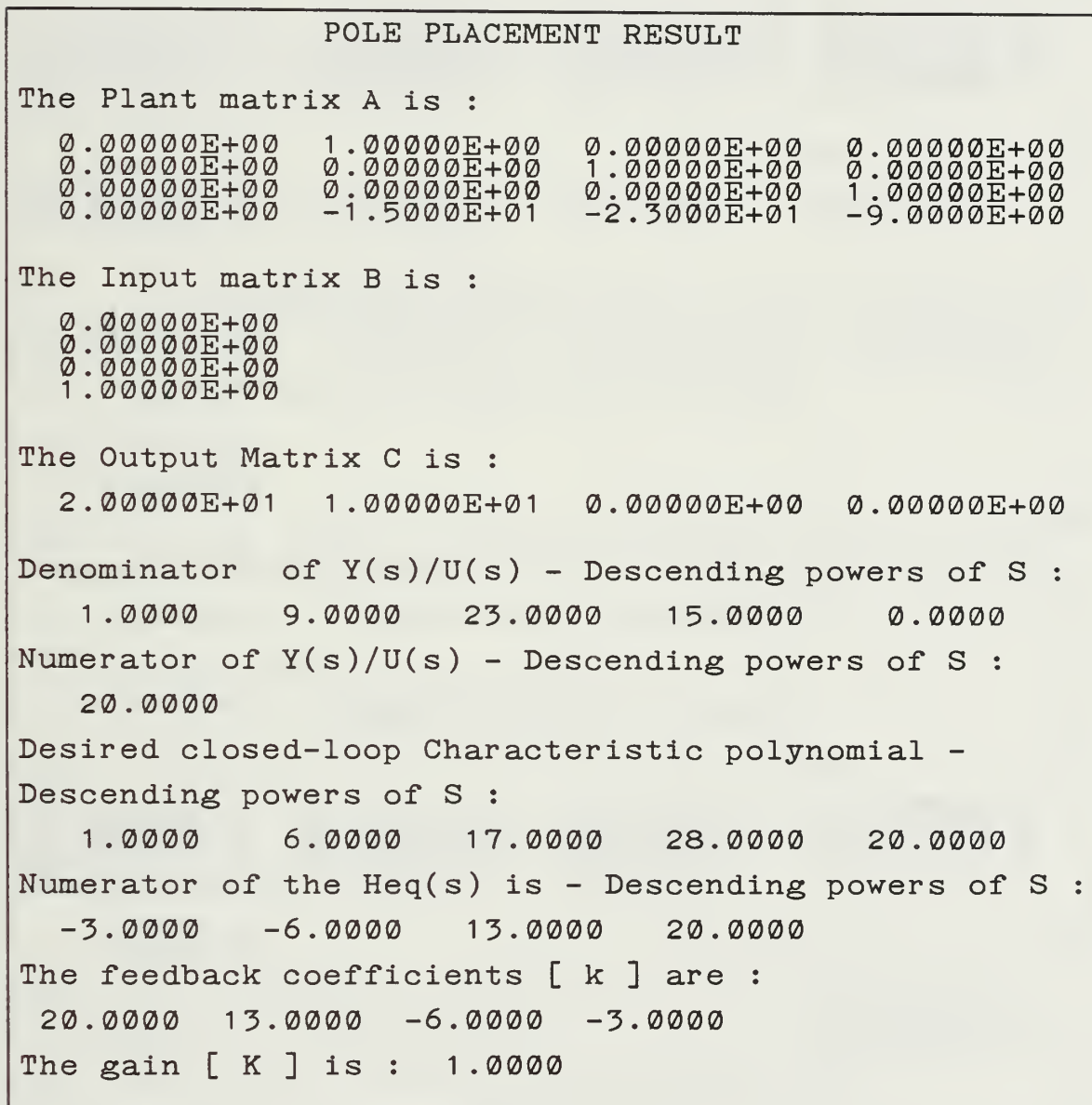


Figure 4.10 Pole placement result for the Luenberger observer design.

Step2

The observability index is determined using the "Observability" option in the same menu. That results is shown in Figure 4.11.

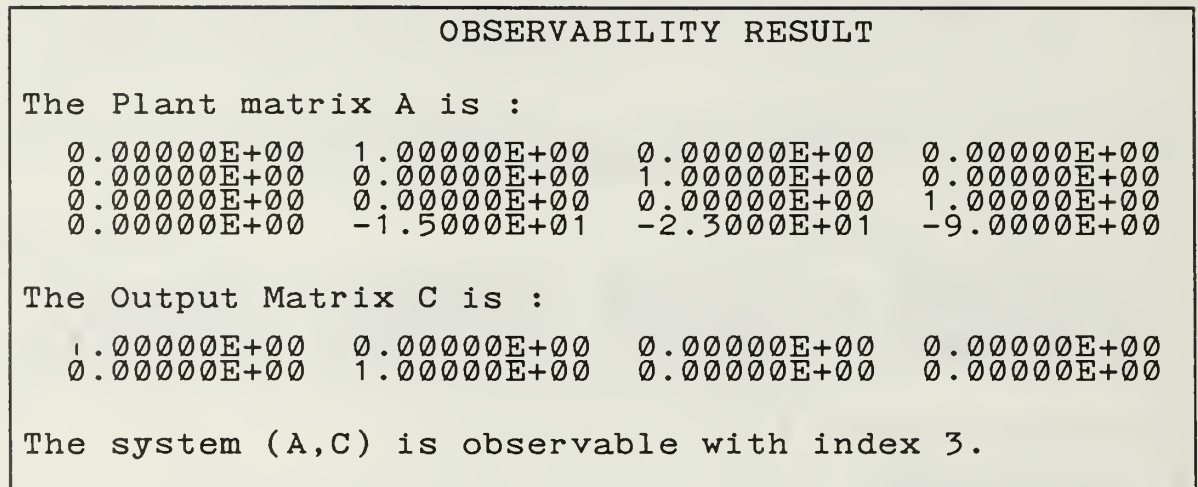


Figure 4.11 Observability result for Luenberger.

Step3

As can be seen from Figure 4.12, the system is completely controllable.

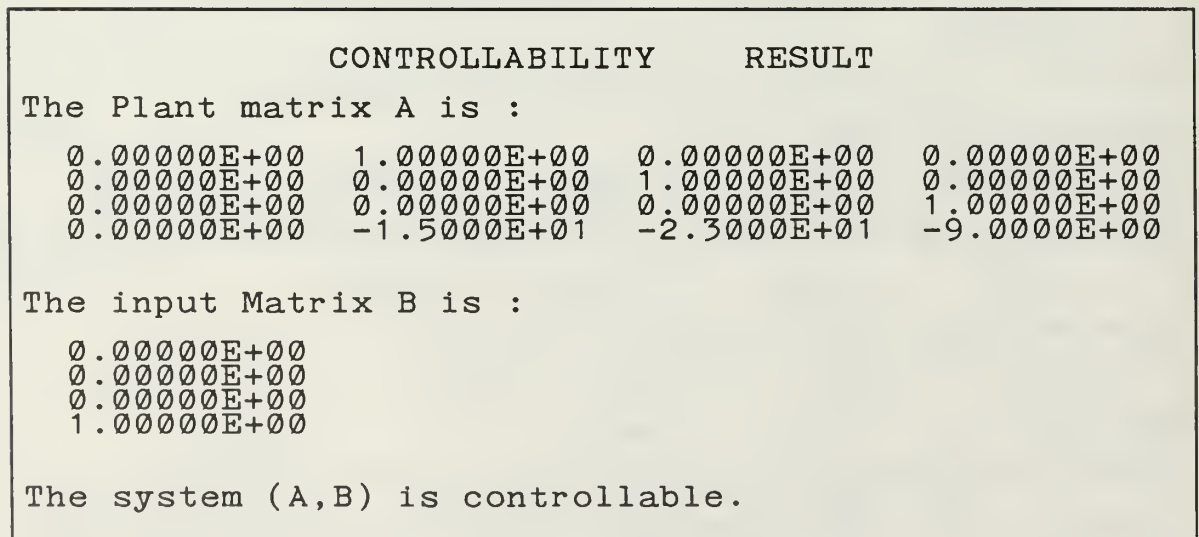


Figure 4.12 Controllability output for Luenberger.

Step4

An observability index $r=3$ (result taken from observability output) allows us to design an observer of order equal to or greater than $(r-1)=2$. For this

```

                                LUENBERGER OBSERVER RESULT

The plant matrix A is :
0.000000E+00  1.000000E+00  0.000000E+00  0.000000E+00
0.000000E+00  0.000000E+00  1.000000E+00  0.000000E+00
0.000000E+00  0.000000E+00  0.000000E+00  1.000000E+00
0.000000E+00 -1.50000E+01 -2.30000E+01 -9.00000E+00

The input matrix B is :
0.000000E+00
0.000000E+00
0.000000E+00
1.000000E+00

The output Matrix C is:
1.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00
0.000000E+00  1.000000E+00  0.000000E+00  0.000000E+00

The desired feedback coefficients are :
2.000000E+01
1.300000E+01
-6.000000E+00
-3.000000E+00

The Observer characteristic polynomial coefficients
in ascending powers of S
1.400000E+01  7.500000E+00  1.000000E+00

The F Matrix is :
-7.50000E+00  1.000000E+00
-1.40000E+01  0.000000E+00

The G1 Matrix is :
8.550000E+01  2.92500E+01
0.000000E+00  0.000000E+00

The G2 Matrix is :
-3.00000E+00
-1.50000E+00

The output feedback coefficients are :
2.000000E+01  8.500000E+00

The compensator feedback coefficients are :
1.000000E+00  0.000000E+00
```

Figure 4.13 The Luenberger observer design output.

example, we want to design observer eigenvalues of -3.5 and -4.0.

Step5

The program results for all this input data is shown in Figure 4.13. From results, the complete system can be described as:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & -15. & -23. & -9. \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u(t)$$

$$\dot{\mathbf{z}}(t) = \begin{bmatrix} -7.5 & -1.0 \\ -14. & 0.0 \end{bmatrix} \begin{bmatrix} z_3(t) \\ z_4(t) \end{bmatrix} + \begin{bmatrix} 85.5 & 29.25 \\ 0.0 & 0.0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} -3.0 \\ -1.5 \end{bmatrix} u(t)$$

$$u(t) = 1.0 r(t) - [20.0 \quad 8.5] \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} - [1.0 \quad .0] \begin{bmatrix} z_3(t) \\ z_4(t) \end{bmatrix}$$

F. DESIGN OF OPTIMAL CONTROL

The system considered is described by the following state variable equation:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \mathbf{x}(t) + \mathbf{B} u(t) \quad (4-26)$$

where

$\mathbf{x}$ = state vector
 u = control input
 $\mathbf{A}$ = plant matrix

B = input matrix

The design of optimal control will minimize the following cost function:

$$J(N) = 1/2 [X^T(N) Q X(N) + \sum_{k=0}^{N-1} [X(k) Q X(k) + R U^2(k)]] \quad (4-27)$$

where the following are defined

Q = noise covariance matrix

N = time intervals over which the SUM is made

R = scalar random input

and X^T means transpose of X.

The physical interpretation of $J(N)$ is this: we wish to keep the state near zero without excessive control energy expenditure. The input parameters of the design of optimal control are entered in the beginning of the program. This screen can be seen in Figure 4.14. After entering these parameters and the Q matrix, then the program calculates the feedback gain matrix which, when multiplied by the state vector, yields a scalar control. In the program procedures, the following equations were derived using dynamic programming, starting at the terminal time and working backwards.

$$P(k) = PSIT(k) * P(k-1) * PSI(k) + Q + GT(k) * R * G(k), \quad P(0) = 0 \quad (4-28)$$

$$PSI(k) = FI + GAMMA * GT(k), \quad PSI(0) = 0 \quad (4-29)$$

$$GT(k) = -[GAMMAT * P(k-1) * FI] / [GAMMAT * P(k-1) * GAMMA + R], \quad GT(0) = 0 \quad (4-30)$$

```

*** Design of optimal control procedure ***
=====
Input number of time intervals for SUM procedure? 40
What is your sample interval? 0.1
What is the value of scalar R? 2.0

For the following options which cost function
do you want? 0

COST=terminal+fuel+trajectory or
COST=terminal+trajectory <0>
COST=terminal+fuel or COST=terminal <1>
where

terminal= 1/2 XT(N) Q X(N)
trajectory= 1/2  $\sum_{k=0}^{N-1} X(k) Q X(k)$ 
fuel = 1/2  $\sum_{k=0}^{N-1} R U^2(k)$ 

Press <ESC> to change it!,
Then type your input with <ENTER> key

```

Figure 4.14 Optimal control parameters screen.

For simplicity in programming, the following definitions are defined:

$$\text{Terminal} = 1/2 \text{ XT(N) } * \text{ Q } * \text{ X(N)} \quad (4-31)$$

$$\text{Trajectory} = 1/2 \sum_{k=0}^{N-1} \text{ X(k) } * \text{ Q } * \text{ X(k)} \quad (4-32)$$

$$\text{Fuel} = 1/2 \sum_{k=0}^{N-1} \text{ R } * \text{ U}^2(\text{k}) \quad (4-33)$$

EXAMPLE 4.3

Given the system equation and parameters described below find the discrete steady state gains for a sample interval 0.1, scalar R is 1.0 and number of time intervals is 40.

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 5 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \quad (4-34)$$

The graphic result of the program is shown in Figure 4.15 and Figure 4.16, the numerical output is in the Figure 4.17.

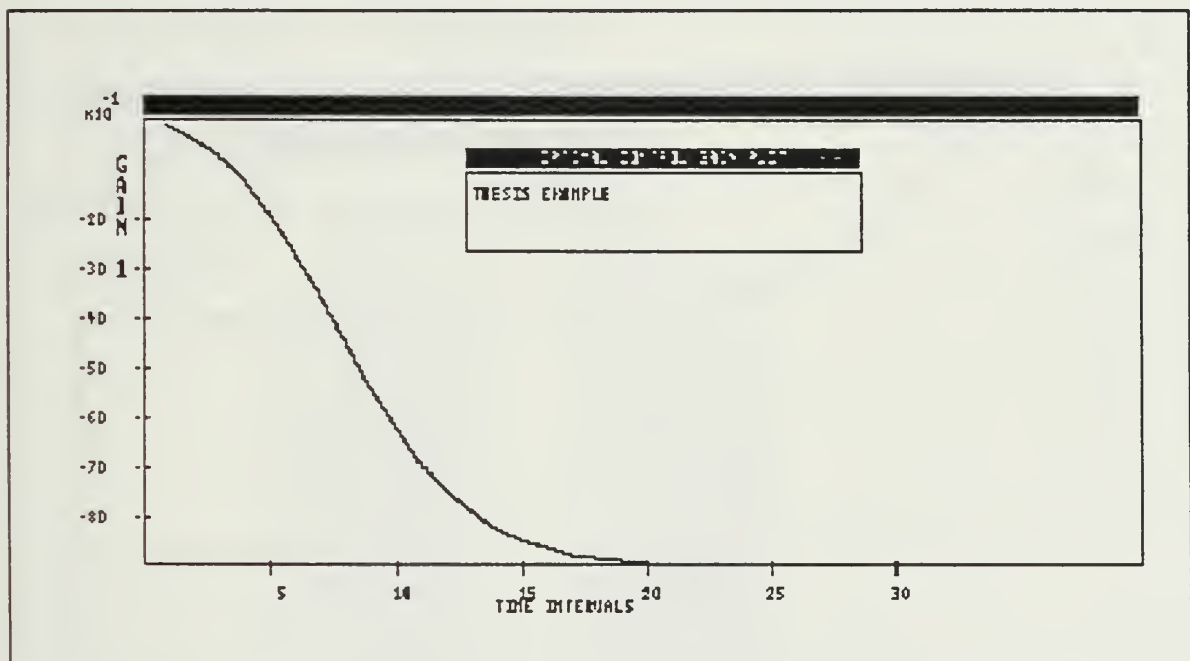


Figure 4.15 Optimal control graphic output #1.

G. MATRIX MATHEMATICS MENU

This option is used to get various calculations with the plant matrix A of a given linear control state variable system. The selection from the SVS main menu

brings the matrix mathematics menu. This second menu consist of following options:

- (1) The determinant of A matrix
- (2) The inverse of A matrix
- (3) The characteristic polynomial of A matrix
- (4) The Eigenvalues of A matrix

Before selecting this option, the user must select "Input/Change Plant Matrices" option to enter the plant matrices.

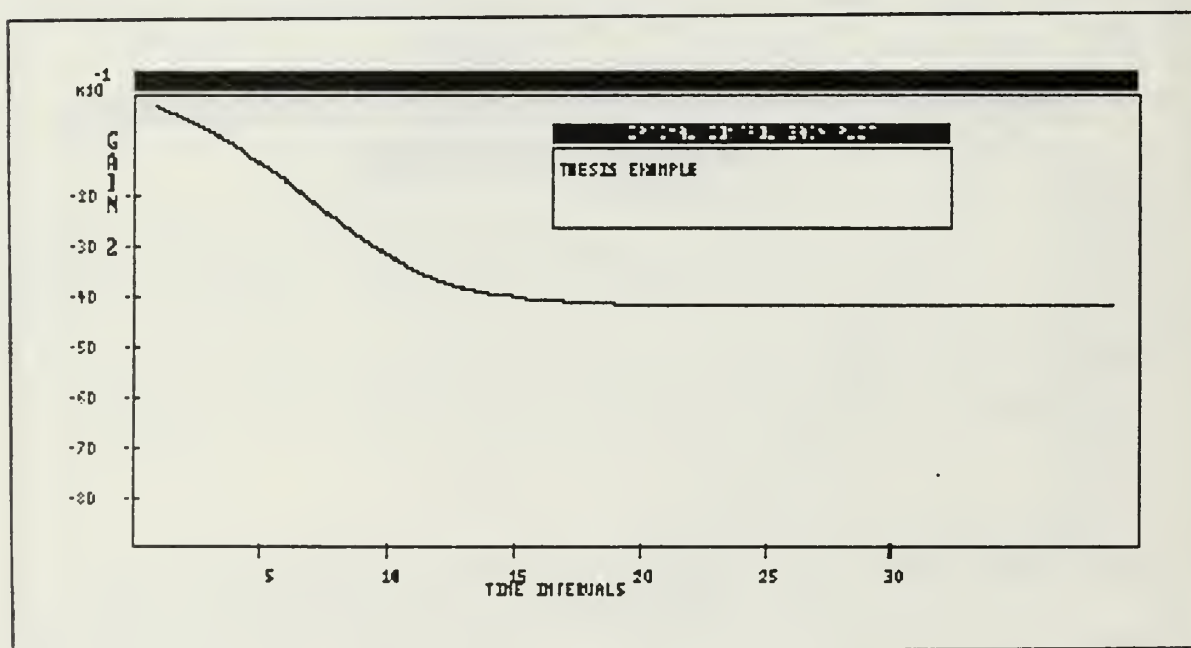


Figure 4.16 Optimal control graphic output #2.

OPTIMAL CONTROL RESULT

The order of the system is : 2
The number of time intervals is : 1.0E-01
The scalar R is : 1.0000
The sample interval is : 40

The A matrix is:

```
0.000000E+00  1.000000E+00
5.000000E+00  0.000000E+00
```

The B matrix is:

```
0.000000E+00
1.000000E+00
```

The Q matrix is:

```
1.000000E+00  0.000000E+00
0.000000E+00  2.000000E+00
```

The FI matrix is:

```
1.02510E+00  1.00835E-01
5.04177E-01  1.02510E+00
```

The GAMMA matrix is:

```
5.02087E-03
1.00835E-01
```

MINIMIZATION OVER ALL STAGES

(stages)	GT:2	GT:2
1	-1.047E-01	-2.031E-01
2	-3.243E-01	-4.211E-01
3	-6.722E-01	-6.665E-01
4	-1.164E+00	-9.495E-01
5	-1.809E+00	-1.275E+00
6	-2.597E+00	-1.640E+00
7	-3.494E+00	-2.031E+00
8	-4.437E+00	-2.424E+00
9	-5.355E+00	-2.794E+00
10	-6.185E+00	-3.120E+00
11	-6.888E+00	-3.390E+00
12	-7.452E+00	-3.603E+00
13	-7.886E+00	-3.765E+00
14	-8.210E+00	-3.883E+00
15	-8.447E+00	-3.969E+00
16	-8.617E+00	-4.030E+00
17	-8.738E+00	-4.073E+00
18	-8.824E+00	-4.104E+00
19	-8.884E+00	-4.125E+00
20	-8.927E+00	-4.140E+00
21	-8.957E+00	-4.150E+00
22	-8.978E+00	-4.157E+00
23	-8.993E+00	-4.163E+00
24	-9.004E+00	-4.166E+00
25	-9.011E+00	-4.169E+00
26	-9.016E+00	-4.170E+00
27	-9.020E+00	-4.172E+00
28	-9.023E+00	-4.173E+00
29	-9.025E+00	-4.173E+00

Figure 4.17 Optimal control numerical output.

V. CONCLUSION AND RECOMMENDATIONS

The software, SVS, is written as a teaching learning tool for student use. It can be a nice tool for analysis and design of linear control state variable systems. The program is fully interactive and menu driven. The user does not get lost in the program. The <Q> key always returns to the main menu or CTRL + C key interrupts the program. All options were tested solving several example problems. Hopefully, all "bugs" have been eliminated.

Furthermore, the program can still be easily improved and expanded. These are listed below.

- (1) Adding the ability to handle the KALMAN filtering.
- (2) Add the capability of the program to handle discrete time systems.

APPENDIX A

UTILITI FILES

A set of Turbo Pascal input and menuing utilities was used widely in the program. These are public domain utilities and was copied from the LCS-CAD source diskettes. These utilities, called TURBO-UT.PAS were written by:

Donald R. Ramsey
Larry Romero
727 Bunker Hill #70
Houston, Texas 77029

and distributed through the public domain. The describing documentation are presented on reference 4.

APPENDIX B

PROGRAM LISTING

Appendix B is a listing files of the Turbo Pascal source code. These files make up the major modules of the SVS program. In general, most of the driver programs and including files are listed in the following pages. The source modules from the Borland International Turbo Graphix Toolbox and the input/output utility routines are not listed.

```

Program State Variable System(input,output);
(* ***** *)
(* The following include files contain procedures *)
(* to handle graphics call from the main procedures *)
(* of this program. These include files are a part *)
(* of Borland International's Turbo Graphix Toolbox *)
(* Which is a commercially available product. *)
(* ***** *)

{$I Typedef.SYS}      {type & variable decleration}
{$I Box.INC}           {draw the box for the main menu}

(* ***** *)
(* These are utility procedures and functions to *)
(* help input. They are public domain programs. *)
(* ***** *)

{$I Ut-Mod01.INC}      {I/O procedures}
{$I Ut-Mod02.INC}

Procedure MainMenu;
var
  I,Tab                               : Integer;
  Help1File,InputFile,MatrixFile,
  plotfile,ControlFile,ObserFile,
  LuenbergFile,PoleFile,OptimalFile  : File;
  Description                         : Str80;
  Okchoices                           : Set Of Char;

procedure ProgramExit;
      (Displays warning about program end)
begin
  ClrScr; Highvideo;
  Msg('Have you wanted ending the SVS program? ( Y/N )',10,10);
  LowVideo;
  Repeat
    option;
  until (Ch in ['Y','N']) then Beep(900,350)
End;

Procedure MenuItem(pick:char;description:str80;
      color:integer);
      ( allows easy selection of main menu colors)
Begin
  TextColor(color);
  Write(' ',Tab,'); TextColor(white ); Write(pick);
  TextColor(color); Writeln(' ',description);
End;

Begin ( Main Menu )
  ClrScr;
  TextColor(lightblue);
  Msg('Naval Postgraduate School',Ismail);
  UNLU(7,24);
  GoToXY(21,4); TextColor(white);
  Writeln(' *** SVS MAIN MENU *** ');
      { show main menu }
  Writeln(' ');
  Tab:=23;
  MenuItem('I','Input / Change Plant Matrix Menu',red);
  Writeln;
  MenuItem('G','Graphics Menu',lightgray);
  MenuItem('C','Controllability',yellow);
  MenuItem('O','Observability',yellow);
  MenuItem('L','Luenberger Observer Design',yellow);

```

```

MenuItem('D','Design of Optimal Control',yellow);
MenuItem('P','Pole Placement',yellow);
Writeln;
MenuItem('M','Matrix Mathematics Menu',lightgray);
Writeln;
MenuItem('H','Help',lightmagenta);
MenuItem('Q','Quit the program',lightmagenta);
Box;
if not (block1 in [1..20]) then
    OKchoices:=['I','H','Q'];
else
    OKchoices:=['I','C','O','D','M','P',
                'G','L','H','Q'];

Repeat      (sets legal choices depending on user)
Option;
If not (ch in okchoices) then
Begin
    TextColor(red);
    If block1 <= 0 then
    Begin
        Beep(900,350);
        Msg('WARNING: First INPUT ,HELP or QUIT!',
            1,25);
    End;
End; TextColor(white);
Until Ch in Okchoices;
Case Ch of
'I': Begin
        Assign(InputFile,'Input.com');
        Execute(InputFile);
    End;
'M': Begin
        Assign(MatrixFile,'Matrix.com');
        Execute(MatrixFile);
    End;
'G': Begin
        Assign(Plotfile,'plot.com');
        Execute(plotfile);
    End;
'C': Begin
        Assign(ControlFile,'Control.chn');
        Chain(ControlFile);
    End;
'O': Begin
        Assign(ObserFile,'Obser.chn');
        Chain(ObserFile);
    End;
'L': Begin
        Assign(LuenbergFile,'Luenberg.chn');
        Chain(LuenbergFile);
    End;
'D': Begin
        Assign(optimalfile,'optimal.chn');
        Chain(optimalfile);
    End;
'H': Begin
        Assign(Help1File,'Help1.chn');
        Chain(Help1File);
    End;
'P': Begin
        Assign(PoleFile,'Pole.com');
        Execute(PoleFile);
    End;
'Q': Begin
        ProgramExit;
        if (ch='Y') or (ch = 'y') then
            Exit := True;
    End;

```

```
End; {case}
End; {main menu}

(* ***** *)
(*               Program Execution part               *)
(* ***** *)
Begin {svs}
  ClrScr; {clear screen}
  Exit := False ;{initialize boolean}
  Repeat
    if block1 <> 1 then
      Begin
        Block1 := 0;
      End;
      MainMenu; { call main menu until the user
                want to exit}
  Until Exit= True;
  Set_Cap_Num(' ',' ',' '); Say Cap_Num;
  {set caps,insert,and num lock off}
  Block1 := 0; {reset problem}
End. {svs}
```

```

Program input(input,output);
(* This program allows the user to enter,change,save
and retrieve the problem for the whole options on the
other menues. *)

```

```

{$I Typedef.SYS} {comman type & variable definitions}

```

```

{$I Ut-mod01.INC} {I/O utility routines}

```

```

{$I Ut-mod02.INC}

```

```

{$I Ut-mod03.INC}

```

```

{$I Box.INC} {draws menu box}

```

```

var
  inputdatfile,savefile,retrievefile,
  changefile,help2file      : file;

```

```

Procedure InputMenu;

```

```

var i,Tab      : Integer;
    Okchoices  : Set Of Char ;
    Finished   : Boolean;

```

```

Procedure MenuItem(Pick:Char;Description:Str80;
                   Color:Integer);
{displays menu items in color }

```

```

Begin
  TextColor(color);
  Write(' ',:tab,{' '); TextColor(white); Write(pick);
  TextColor(color); Writeln(' ',Description);
End;

```

```

Begin
  ClrScr; TextColor(white); Finished := False;
  GoToXY(20,4); retriev:=true;
  Write('*** INPUT / CHANGE MENU *** ');
  for i := 1 to 4 do writeln(' ');
  Tab := 18;
  MenuItem('I','Input Plant Matrices ',red);
  MenuItem('C','Change Current Plant Matrices',yellow);

```

```

  Writeln;
  MenuItem('S','Save Plant Matrices to Disk
             File',yellow);
  MenuItem('L','Load Plant Matrices From Disk
             File',yellow);

```

```

  Writeln;
  MenuItem('H','Help',lightmagenta);
  MenuItem('Q','Quit to SVS Main Menu ',lightmagenta);

```

```

  Box; TextColor(white);

```

```

  Set Cap_num('C',' ',' ');say_cap_num; gotoxy(40,22);
  if block1 <> 1 then OKchoices := ['I','L','H','Q'];
  else OKchoices := ['I','C','H','S','L','Q'];

```

```

  Repeat {wait for user to input a keyprees}
    option;
    if not (ch in OKchoices) then
      begin
        beep(900,350);TextColor(red);
        if block1 <> 1 then
          begin
            msg('WARNING: First INPUT,RETRIEVE,HELP or
                QUIT!',1,25);
          end;
        end;
        TextColor(white);
      until Ch in OKchoices;

```

```
case ch of
  'I': begin
    Assign(inputdatfile,'inputdat.chn');
    chain(inputdatfile);
  end;
  'L': begin
    Assign(retrievefile,'retrieve.chn');
    Chain(retrievefile);
  end;
  'S': begin
    Assign(savefile,'save.chn');
    Chain(savefile);
  end;
  'C': Begin
    Assign(changeFile,'change.chn');
    Chain(changeFile);
  End;
  'H': Begin
    Assign(help2file,'help2.chn');
    Chain(help2file);
  End;
  'Q': begin
    Assign(svsFile,'svs.com');
    Execute(svsFile);
  end;
End; {case}
end;

{ Execution the input/change menu  program. }

Begin {main program}
  drive:='C';
  repeat
    InputMenu;
  Until Finished = True;
End. {main program}
```

```

Program input_data(input,output);
{To allow the user to enter the A,B,C matrix from the
this program}
label 13;label 10;label 11; {label decleration for the
GOTO statement}

{$I Typedef.sys} {comman variable declerations}
{$I Ut-mod01.inc} {I/O routines}
{$I Ut-mod02.inc}

var Ans,Cont           : Char;
    Out                : Text;
    Stepping,Step,Steps,
    Temp1,Temp2,i,j    : Integer;
    Result             : Real;
    inputfile          : file;

Begin {input_data}
13:ClrScr;Writeln;
block1:=1; textcolor(yellow);
Write('Enter the degree of the plant: ');
Readln(size);
if (size <= 0) or ( size > 10) then
begin
    beep(900,350); goto 13;
end;
for steps:=1 to 10 do {initialize}
    for stepping := 1 to 10 do
        A1A[steps,stepping] := 0.0;

Writeln('Enter the elements of the A Matrix ');
writeln;
for steps := 1 to size do
Begin
    for stepping := 1 to size do
    Begin
        Write('A(',steps,',',stepping,') = ');
        Readln(A1A[steps,stepping]);
    End;Writeln;
End;
Repeat
    ClrScr; Writeln;
    Writeln('The A Matrix is : '); writeln;
    for steps :=1 to size do
    Begin
        for stepping := 1 to size do
        Begin
            Write(' ', A1A [steps,stepping]:11);
        End; Writeln;
    End; writeln; {prompt to any changes}
    Write('Do you want to change any element of');
    write(' the Matrix ? ( Y / N ) ');
    Read(Kbd,Ans); writeln; {allows user to change
                             entered data}
    if ( Ans = 'Y') or ( Ans = 'y') then
    Begin
        write('Input the row to change : '); readln(i);
        write('Input the column to change : ');
        readln(j);writeln;
        write('A(',i,',',j,')= ');readln(result);
        A1A[i,j]:=result;
    End;
Until Ans in ['N','n'];

10:ClrScr;writeln;
{user inputs B matrix elements from the keyboard}

```

```

write('How many inputs do you have ? ');
readln(ni);writeln;
if ( ni < 1 ) or ( ni > size ) then
begin
  beep(900,350);goto 10;
end;
for steps := 1 to 10 do
  for stepping:=1 to 10 do
    B[steps,stepping] := 0.0;
writeln('Enter the elements of the B Matrix ');
writeln;
for steps := 1 to size do
  Begin
    for stepping:=1 to ni do
      begin
        Write('B(',steps,',',stepping,') = ');
        Readln( B [ steps,stepping ] );
      end;writeln;
    End; writeln;
  Repeat
    ClrScr; Writeln;
    Writeln('The B Matrix is: '); writeln;
    for steps :=1 to size do
      Begin
        for stepping:=1 to ni do
          begin
            Write(' ',B[steps,stepping]:11);
          end; writeln;
        End; Writeln;
        Write('Do you want to change any element of');
        write(' the B Matrix ? ( Y / N ) ');
        Read(Kbd,Ans);
        writeln; {allows user to change B matrix element}
        if ( Ans ='Y') or ( Ans ='y') then
          Begin
            write('Input the row to change : ');readln(i);
            write('Input the column to change : ');
            readln(j); writeln;
            write('B(',i,',',j,')= ');readln(result);
            B[i,j]:=result;
          End;
        Until Ans in ['N','n'];
      11:ClrScr; Writeln;
      {user inputs output data from keyboard}
      write('How many outputs do you have ? ');
      readln(no);
      if ( no < 1 ) or ( no > size ) then
      begin
        beep(900,350);goto 11;
      end;
      for steps:= 1 to 10 do
        for stepping:=1 to 10 do
          C[steps,stepping]:=0.0;
        writeln;
        Writeln('Enter the elements of the C Matrix ');
        Writeln;
        for stepping := 1 to no do
          Begin
            for steps :=1 to size do
              begin
                Write('C(',stepping,',',steps,') = ');
                Readln( C[stepping,steps] );
              end; writeln;
            End; writeln;
          Repeat
            ClrScr; Writeln;
            Writeln('The C matrix is : '); writeln;

```

```
for stepping := 1 to no do
Begin
  for steps:=1 to size do
    begin
      Write(' ',C[stepping,steps]:11);
    end;writeln;
  End; Writeln;
  Write('Do you want to change any element of');
  write(' the C Matrix ? ( Y / N ) ');
  Read(Kbd,Ans); writeln;
  if ( Ans ='Y') or ( Ans ='y') then
  Begin
    write('Input the row to chnage : ');readln(i);
    write('Input the column to change : ');
    readln(j); writeln;
    write('C( ',i,',',j,')= ');readln(result);
    C[i,j]:=result;
  End;
  Until Ans in ['N','n'];
  Assign(inputfile,'input.com');
  {re-execute the input/change menu program}
  Execute(inputfile);
End.{input_data}
```

```

Program change data(input,output);
(* ***** *)
(* This program allows the user to change A,B, C *)
(* matrices and their matrix order. *)
(* ***** *)
label 11;label 12;label 14;

{$I Typedef.sys}    {common type & variable decleration}
{$I Ut-mod01.inc}    {I/O utilities }
{$I Ut-mod02.inc}

var  Ans,temp,inputtype      : Char;
     Stepping,Step,Steps,i,j : Integer;
     Result                  : Real;
     inputfile               : file;
     result_size             : integer;

Begin
  clrscr;writeln; gotoxy(1,22);
  invideo('Press <ESC> to change it!, ');
  gotoxy(1,23);
  invideo('Then input your choice with <ENTER> key');
  gotoxy(1,2);
  textcolor(lightblue);
  writeln('*** Change Current Plant Matrices
          Procedure ***');
  TextColor(yellow);
  writeln('=====');

  Msg('Which matrix do you want to change ? ',1,6);
  textcolor(lightmagenta);
  msg('          PLANT (A) ',39,6);
  msg('          INPUT (B) ',1,7);
  msg('          OUTPUT (C) ',1,8);
  repeat
    input('A','A',50,6,2,true,F1,F10);
    temp:=copy(answer,1,1);
    if not (temp in ['A','B','C']) then beep(900,350);
  until temp in ['A','B','C'];
  inputtype:= temp;

  case inputtype of
    'A':begin
      14:ClrScr;
      writeln('The A matrix is :');writeln;
      for steps:=1 to size do
        begin
          for stepping:=1 to size do
            begin
              write(' ',A1A[steps,stepping]:11);
            end;writeln;
          end;writeln;
      Write('The order of the system is:',
            size,',');
      write(' Change ? (Y/N)');
      repeat
        read(kbd,ch);
      until ch in ['Y','N','y','n'];
      if (ch='Y') or (ch='y') then
        begin
          writeln;
          write('The order of the system is :');
          readln(result_size);
          if (result_size<1) or (result_size>10)
          then begin
            beep(900,350); goto 14;
          end;
        end;
    end;
  end;

```

```

        end;
        size:=result_size;
    end;
    repeat
        clrscr;writeln;
        Writeln('The A Matrix is : ');writeln;
        for steps :=1 to size do
            Begin
                for stepping := 1 to size do
                    Begin
                        Write(' ',A[A[steps,stepping]:11]);
                    End; Writeln;
                End;Writeln;
                Write('Do you want to change any element?
                    ( Y / N )');
                Read(Kbd,Ans);writeln;
                if not (Ans in ['N','n']) then
                    if (Ans ='Y') or (Ans ='y') then
                        Begin
                            write('Input row to change : ');
                            readln(i);
                            write('Input column to change : ');
                            readln(j); writeln;
                            write('A( ,i , ,j , )=');
                            readln(result);
                            A[A[i,j]:=result;
                        End;
                    Until Ans in ['N','n'];
                end;
            'B':begin
                11:ClrScr;
                writeln('The B matrix is :');writeln;
                for steps:=1 to size do
                    begin
                        for stepping:=1 to ni do
                            begin
                                write(' ',B[steps,stepping]:11);
                            end;writeln;
                        end;writeln;
                        Write('The number of input is : ',ni,' ,
                            Change ? (Y/N)');
                        repeat read(kbd,ch)
                        until ch in ['Y','N','y','n'];
                        if (ch='Y') or (ch='y') then
                            begin
                                writeln; {allow the user to change number
                                    of input}
                                write('The number of input is:');
                                readln(result_size);
                                if (result_size<1) or (result_size>10)
                                    then begin
                                        beep(900,350); goto 11;
                                    end;
                                ni:=result_size;
                            end;
                        end;
                    repeat
                        clrscr;writeln;
                        Writeln('The B Matrix is : ');writeln;
                        for steps :=1 to size do
                            Begin
                                for stepping := 1 to ni do
                                    Begin
                                        Write(' ',B[steps,stepping]:11);
                                    End; Writeln;
                                End;Writeln;
                                {prompt the user for changes on B matrix}

```

```

Write('Do you want to change any element
? ( Y / N )');
Read(Kbd,Ans);writeln;
if not (Ans in ['N','n']) then
if ( Ans = 'Y') or ( 'Ans ='y') then
Begin
write('Input row to change : ');
readln(i);
write('Input column to change : ');
readln(j);
writeln;write('B(',i,',',',',j,')=');
readln(result);
B[i,j]:=result;
End;
Until Ans in ['N','n'];
end;
'C':begin
12:ClrScr;
writeln('The C matrix is :');writeln;
for steps:=1 to no do
begin
for stepping:=1 to size do
begin
write(' ',C[steps,stepping]:11);
end;writeln;
end;writeln;
{allow the user to change number of outputs}
Write('The number of output is : ',no,'');
write(' Change ? (Y/N)');
repeat read(kbd,ch)
until ch in ['Y','N','y','n'];
if (ch='Y') or (ch='y') then
begin
writeln;
write('The Number of Output is :');
readln(result_size);
if (result_size<1) or (result_size>10)
then begin
beep(900,350); goto 12;
end;
no:=result_size;
end;
repeat
clrscr;writeln;{show C matrix elements}
writeln('The C Matrix is : ');writeln;
for steps :=1 to no do
Begin
for stepping := 1 to size do
Begin
write(' ',C[steps,stepping]:11);
End; Writeln;
End;Writeln;
{prompt the user for changes on C matrix }
Write('Do you want to change any element
? ( Y / N )');
Read(Kbd,Ans);writeln;
if not (Ans in ['N','n']) then
if ( Ans = 'Y') or ( 'Ans ='y') then
Begin
write('Input row to change : ');
readln(i);
write('Input column to change : ');
readln(j);
writeln;write('C(',i,',',',',j,')=');
readln(result);
C[i,j]:=result;
End;

```

```
        Until Ans in ['N','n'];
    end;
end;
Assign(inputfile,'input.com'); {re-execute input
                                program}
Execute(inputfile);
End.{change}
```

```

Program save_data;

{$I Typedef.SYS}
{$I Ut-mod01.INC}
{$I Ut-mod02.inc}

var
  filename      : str20;
  blockfile     : text;
  i,j           : integer;
  inputfile     : file;
begin {save_data}
  clrscr; HighVideo;
  {let user to change drive if necessary}
  Msg('*** Your Data Drive is      . Press <esc> to change
      it!***',10,11);
  repeat
    input('A',copy(drive,1,1),32,11,2,true,F1,F10);
    ch:= copy(answer,1,1);
    if not(ch in ['A','B','C','D','E']) then
      beep(350,150);
  until ch in ['A','B','C','D','E'];
  Drive := concat(ch,',');
  clrscr;          {prompt for filename to store data}
  msg('Input name of file to save data in File',1,10);
  writeln; writeln;
  writeln('*** Your DATA disk must be in drive
  ',Drive,' ***');
  input('A','',45,10,8,true,F1,F10);
  filename:= concat(Drive,copy(answer,1,8),'.svs');

  Assign(Blockfile,filename); {Open file }
  rewrite(Blockfile);
  Writeln(Blockfile,size);
  Writeln(Blockfile,ni);
  Writeln(Blockfile,no);
  for i:= 1 to size do
    for j:=1 to size do
      write(blockfile,A1A[i,j]);
  for i:=1 to size do
    for j:= 1 to ni do
      writeln(blockfile,B[i,j]);
  for i:=1 to no do
    for j:=1 to size do
      write(blockfile,C[i,j]);

  close(Blockfile); {close the file}
  Assign(inputfile,'input.com');{re-execute
                                input/change menu program}
  Execute(inputfile);
end.{save_data}

```

```

Program retrieve_data;
{$I Typedef.SYS} -{variable declerations}

{$I Ut-mod01.INC} {I/O routines}
{$I Ut-mod02.INC}
{$I Ut-mod03.INC}

{$I Directry.INC} {shows available data files}

var
  readfile      : text;      {The text file user will use}
  filename      : str20;
  readerror     : boolean;
  linecounter,i,j : integer;  { A counter for the
                                lines we read}
  inputfile,PlotFile: file;

begin { procedure retrieve_data}
  ClrScr;HighVideo;
  { Allow change of disk drive if desired}
  block1:=1;
  Msg('** Your data drive is . Press <ESC> to change
    it! **',10,11);
  Repeat
    Input('A',copy(drive,1,1),32,11,2,true,F1,F10);
    Ch:= copy(answer,1,1);
    If not(ch in ['A','B','C','D','E']) then
      beep(900,350);
  Until ch in ['A','B','C','D','E'];
  Drive:= concat(ch,':');
  { Call directory to display eligible files}
  Directory(drive,extension,filename,readerror);
  If not(readerror) then
    Begin
      Assign(readfile,filename);
      { Open the file and read contents}
      Reset(readfile);
      Linecounter:=0; {count the lines }
      While not EOF(readfile) do
        {The built-in function EOF returns true}
        Begin{if the end of a file has been reached}
          Linecounter:=linecounter+1;{Count the next line}
          Readln(readfile,size);
          { Read it into variable line }
          Readln(readfile,ni);
          Readln(readfile,no);
          For i:=1 to size do
            For j:= 1 to size do
              Read(readfile,A1A[i,j]);
          For i:=1 to size do
            For j:=1 to ni do
              Readln(readfile,B[i,j]);
          For i:= 1 to no do
            For j:=1 to size do
              Read(readfile,C[i,j]);
          End; Close(readfile);
        End
      Else
        Begin
          Delay(1500);
          { Wait for directory error message and continue}
          Window(1,1,80,25);
          ClrScr;
        End;
      if not (retriev) then
        begin
          Assign(plotfile,'Plot.COM');
          Execute(PlotFile);
        end
    end

```

```
end
else
begin
    Assign(inputfile, 'input.com'); {re-execute
                                   input/change menu program}
    Execute(inputfile);
end;
End. {retrieve_data}
```

```

Program Plot;
{ Program plot contains graphic programs. They are
written by WOOD Roy, JR. and modified for this
program. }

```

```

{$I Typedef.SYS}{type and variable decleration routine}

```

```

{$I Ut-mod01.INC}      {I/O routines }
{$I Ut-mod02.INC}
{$I Ut-mod03.INC}

```

```

{$I Box.INC}           {drawing menu box }

```

```

var
  i,j                                     : Integer;
  help4file,Retrievefile,Nyquistfile,
  timeplotfile,bodefile,rlocifile,rootsfile : File;

```

```

Procedure GraphicsMenu;

```

```

var
  i,Tab                                     : Integer;
  Okchoice                                 : Set Of Char ;
  Finished                                 : Boolean;

```

```

Procedure MenuItem(Pick:Char;Description:Str80;
  Color:Integer);
  {gives easy selection of input menu colors}

```

```

Begin
  TextColor(color);
  Write ('',:tab,{''); TextColor(white); Write(pick);
  TextColor(color); Writeln(')',Description);
End;

```

```

Begin
  ClrScr;TextColor(white);
  Finished := False;GoToXY(20,4);
  Writeln(' *** GRAPHICS MENU *** ');
  {display graphics menu}
  Writeln;
  Tab := 18;
  MenuItem('L','Load Plant Matrices From Disk
  File',red);
  Writeln;
  MenuItem('C','Characteristic Equation Roots',yellow);

  MenuItem('B','Bode Plot ',yellow);
  MenuItem('T','Time Response Plot',yellow);
  MenuItem('N','Nyquist plot',yellow);
  MenuItem('R','Root Locus Plot',yellow);
  writeln;
  MenuItem('H','Help',lightmagenta);
  MenuItem('Q','Quit to SVS Main Menu ',lightmagenta);
  TextColor(green);
  Box; Writeln;
  TextColor(white); retriev:=false;
  GoToXY (40,22);
  repeat { read user choice from keyboard}
  option;
  until ch in ['R','L','B','T','N','H','Q','C'];
  case ch of
    'R':begin
      assign(rlocifile,'rloci.chn');
      chain(rlocifile);
      end;
    'L': Begin
      Assign(retrievefile,'retrieve.chn');
      Chain(retrievefile);
      end;
  end;

```

```
'T': begin
    Assign(timeplotfile,'timeplot.chn');
    Chain(timeplotfile);
end;
'B': begin
    Assign(Bodefile,'Bode.chn');
    Chain(bodefile);
end;
'N': begin
    Assign(nyquistfile,'Nyquist.chn');
    Chain(Nyquistfile);
end;
'C': Begin
    Assign(rootsfile,'roots.chn');
    Chain(rootsfile);
End;
'H': Begin
    Assign(Help4File,'Help4.CHN');
    Chain(Help4File);
End;
'Q': begin
    Assign(svsFile,'svs.com');
    Execute(svsFile);
end;
End;
End;
Begin {plot}
    drive:='C';
    {initialize drive selection for load procedure}
    repeat
        GraphicsMenu; {repeadely call graphics menu until
                        user selects to quit}
    Until Finished = True;
End. {plot}
```

{ BODE PLOT is the driver program for the Bode plotting routines. It simply invokes Bode plot, when finished, returns back to the graphics menu. }

Program Bode;

```
{ $I Typedef.sys } {graphics routines}
{ $I Graphix.sys }
{ $I Kernel.sys }
{ $I Windows.sys }
{ $I Polygon.hgh }
{ $I Axis.hgh }
```

```
{ $I Ut-mod01.inc } {I/O routines}
{ $I Ut-mod02.inc }
{ $I Ut-mod03.inc }
```

```
{ $I GrapMenu.inc } {graph options menu}
{ $I PlotBode.inc } {Bode plotting routine}
{ $I Boxuser.inc }
```

```
type
    ary4 = array [1..11] of real;
    ary6 = array [1..21] of integer;
```

```
var
    plotfile :file;
```

```
{ $I Polynom.inc } {Polynomial routine}
{ $I Rootfind.inc } {Polynomial roots finder routine}
{ $I Bodeplot.inc } {Bode routine}
```

```
begin
    BODEPLOT; {call the Bode calculation routine}
```

```
    Assign(plotfile,'PLOT.COM');
    {re-execute the graphics menu routines}
    Execute(plotfile);
```

```
end.
```

```

Program time response;
label 13; {label declaration for GOTO statement}

{$I Typedef.sys}      {graphics routines}
{$I Graphix.sys}
{$I Kernel.sys}
{$I Windows.sys}
{$I Axis.hgh}
{$I Polygon.hgh}

{$I Ut-mod01.inc}      {I/O routines}
{$I UT-mod02.inc}
{$I Ut-mod03.inc}

{$I GrapMenu.inc}      {graph option menu procedure}

type
  ary4      = array [1..11] of real;
  ary6      = array [1..11] of integer;

var
  Psi,Phi,A,Atemp      : ary1s;
  temp, inputtype      : char;
  Offset,Slope,Tmax,
  RowSum,MaxRowSum,t,
  T1,OldMaxRowSum,
  Plottime,Uinput,PhiX,
  hold,Ymax,Ymin,tPlot,
  Amplitude, Freq,y      : real;
  Factorial,Plotindex,
  Nincr,code,i,
  jj,j,l,m,n,kk,cnpoles,sizezero
  DumpGraph,GoodNumbers,
  ClosedLoop,quit      : boolean;
  C1,Xold,Xnext,Gamma
  GraphArray,Inputarray
  List      : ary3s;
  numcoeff,dencoeff,cdencoeff
  kk1      : integer;
  plotfile      : file;

{$I Polynom.inc}      {characteristic equation procedure}
{$I Boxuser.inc}

Procedure PrintGraphData;
{this procedure dumps time-response data to printer}
Begin
  LeaveGraphic;Clrscr;
  repeat
    Textcolor(white); gotoxy(20,10);
    writeln(' *** PROGRAM OUTPUT OPTIONS *** ');
    gotoxy(20,13);
    writeln('<P> Printer output ');
    Textcolor(yellow); gotoxy(20,14);
    writeln(' Check Your Printer! ');
    Textcolor(white); gotoxy(20,15);
    writeln('<F> List to File name ');
    gotoxy(20,16);
    writeln(' on the current drive ');
    gotoxy(20,17);
    writeln('<Q> Quit ');
    gotoxy(42,15);textcolor(yellow);write('"TIME.RES"');

    gotoxy(28,17);
    read(kbd,ch);
    If (ch='F') or (ch='f') or (ch='P') or (ch='p')
    then

```

```

begin
  if (ch = 'F') or (ch = 'f') then
  begin
    gotoxy(24,15);textcolor(red);
    write('PRINTING:');
    assign(list,'Time:RES');
    rewrite(list);
  end
  else
  begin
    gotoxy(24,13);textcolor(red);
    write('PRINTING:');
    assign(list,'LST:');
    rewrite(list);
  end;
  writeln(list,'                      TIME RESPONSE PLOT
                                RESULT ');
  writeln(list);
  writeln(list,'          TIME (Sec)                      R          Y
                                (input)',(output));

  writeln(list,'-----');
  writeln(list);
  for i:=1 to 200 do
  writeln(list,'          GraphArray[i,1]:10:5,',
                                GraphArray[i,2]:12:4,',
                                InputArray[i,2]:12:4);
end;
until ch in ['Q','q'];
EnterGraphic;
swapscreen;
close(list);
end;

Procedure Matrix_Mult(Matrix1,Matrix2:ary1s;
                      var AnswerMatrix:ary1s;
                      Order:integer);
var i,j      : integer;
begin
  for i:=1 to order do
  for j:=1 to order do
    AnswerMatrix[i,j] := 0;
    {initialize the answer matrix}

    for i:= 1 to order do
    for j:= 1 to order do
    for L := 1 to order do
      AnswerMatrix[i,j] := AnswerMatrix[i,j] +
                          Matrix1[i,L]*Matrix2[L,j];
end;

Procedure Scalar_Mult(Matrix1 : ary1s; scalar : real;
                      var AnswerMatrix:ary1s;
                      Order:integer);
var i,j      : integer;
begin
  for i:= 1 to order do
  for j:=1 to order do
    AnswerMatrix[i,j]:= AnswerMatrix[i,j] * scalar;
end;

Procedure Matrix_Vector_Mult(Matrix1 : ary1s;
                              Vector : ary3s;
                              var AnswerVector:ary3s;
                              order:integer);

```

```

var i,j      : integer;
begin
  for i:= 1 to order do
    begin
      hold:= 0;
      for j:= 1 to order do
        hold:= hold + Matrix1[i,j]*Vector[j];
        AnswerVector[i]:= hold;
      end;
    end;
end;

Begin
  Initgraphic;leavegraphic;
  Boxuser;
  Characteristic Equation(A1A,size,Dencoeff);
    {calculating denominator coefficient}

  for i:=1 to size do PSI[i,size]:=B[i,1];
  for jj:=2 to size do
    begin
      for i:=1 to size do
        begin
          j:=size-jj+1;
          kk1:=j+1;
          PSI[i,j]:=Dencoeff[kk1] * B[i,1];
          for l:=1 to size do
            PSI[i,j]:= PSI[i,j] +A1A[i,1] * PSI[l,kk1];
          end;
        end;
      end;
    end;

  for i:=1 to size do
    {calculating numerator coefficients }
    begin
      Numcoeff[i]:= 0.0;
      for j:=1 to size do
        Numcoeff[i]:= Numcoeff[i] + PSI[j,i] * C[1,i];
      end;
    end;
  for i:=1 to size do {calculating numerator order }
    begin
      m:=size+1-i;
      if Numcoeff[m] <> 0.0 then goto 13;
    end;
  13:sizezero:=m-1;
  Clrscr;

  TextColor(lightblue);
  writeln('*** Time Response Plotting
  Parameters ***');
  TextColor(yellow);writeln('=====');
  Msg('What is your input to the system?  STEP (S)
                                     ,1,6);
  Msg('                                RAMP (R)
                                     ,1,7);
  Msg('                                SINE WAVE (W)
                                     ,1,8);
  Msg('                                IMPULSE (I)
                                     ,1,9);

  repeat
    Input('A','S',50,6,2,true,F1,F10);
    temp := copy(answer,1,1);
    if not (temp in ['S','R','W','I']) then
      Beep(350,150);
  until temp in ['S','R','W','I'];
  InputType := temp;
  Msg('What is your input amplitude? ',1,11);
  Input('N','1',35,11,5,true,F1,F10);
  val(answer,Amplitude,code);

```

```

case InputType of
'R': begin
    Msg('What is your DC value? ',1,13);
    Input('N','0',28,13,5,true,F1,F10);
    val(answer,Offset,code);
    msg('What is your slope? ',1,15);
    Input('N','1',23,15,5,true,F1,F10);
    val(answer,slope,code);
end;
'W': begin
    Msg('What is your frequency?
        (rad/sec)',1,13);
    Input('N','',35,13,5,true,F1,F10);
    val(answer,Freq,code);
end;
end;
Msg('Open (O) or Closed (C) Loop Plot ? ',1,17);
repeat
    Input('A','C',40,17,2,true,F1,F10);
    temp := copy(answer,1,1);
    if not (temp in ['O','C']) then beep(350,150);
until temp in ['O','C'];
if temp = 'C' then ClosedLoop:= true
                else ClosedLoop:= false;
Msg('Input your simulation time for the system (99
max)',1,20);
repeat
    Input('N','',55,20,5,true,F1,F10);
    val(answer,Tmax,code);
    if Tmax > 99 then beep(350,150);
until (Tmax <= 99) and (Tmax > 0);
Boxuser;
if ClosedLoop then
begin
    for i:=1 to maxorder do CDenCoeff[i] := 0.0;
    {initialize}
    for i:=1 to sizeZero + 1 do
        CDenCoeff[i] := Numcoeff[i];
    {C.L. denominator equals the sum of open loop
    denominator and O.L. numerator}
    for i:=1 to SIZE + 1 do
        CDenCoeff[i] := CDenCoeff[i] + Dencoeff[i];
    if size > sizezero then CNpoles:= size
        else CNpoles:= SizeZero;
    {Nploes shold always be greater,but to be safe}
end
else
begin
    CNpoles:=Size;
    for i:=1 to size do
        CDenCoeff[i]:=Dencoeff[i];
end;
{ Calculation of new A matrix}
for i:= 1 to CNPoles-1 do
begin
    for j:=1 to CNPoles do
    begin
        if j = i+1 then A[i,j]:= 1.0;
        else A[i,j]:= 0.0;
    end;
end;
end;

```

```

{ Calculation of new C matrix }
for j:= 1 to CNPoles do
begin
  A[CNPoles,j] := -CDenCoeff[j];
end;
for i:= 1 to CNPoles do
begin
  if i > SizeZero + 1 then C1[i]:= 0.0
  else C1[i]:= NumCoeff[i];
  if SizeZero = CNPoles then
    C1[i] := C1[i] + NumCoeff[SizeZero+1]*
    A[CNPoles,i];
end;

{ Selection of sampling time interval }

Nincr := 1000;
T := (Tmax/Nincr);
Atemp := A;
Psi := A; { initialize psi to the value of the
           infinite series after the first }

Scalar_Mult(Psi,T/2,Psi,CNPoles);
for i:= 1 to CNPoles do { two terms I + A*T / 2! }
  Psi[i,i] := Psi[i,i] + 1.0;
Factorial := 2; T1 := T; Oldmaxrowsum := 0.0;
repeat
begin
  Factorial := Factorial * (i+1); T1 := T1 * T;
  Matrix_Mult(A,Atemp,Phi,CNPoles);
  { phi is used at temp }
  Atemp := Phi; { holding matrix to large array }
  Scalar_Mult(Phi,(T1/Factorial),Phi,CNPoles);

  for j:=1 to CNpoles do
  begin
    for m:= 1 to CNpoles do
    begin
      Psi[j,m] := Psi[j,m] + Phi[j,m];
    end;
  end;
  Maxrowsum := 0.0;
  { computes maxrowsum as measure of change in last }
  for j:= 1 to CNpoles do { series term to be added. }
  begin
    rowsum := 0.0;
    for m:=1 to CNPoles do
      rowsum := rowsum + Psi[j,m]; writeln;
    if rowsum > maxrowsum then maxrowsum := rowsum;
  end;
  if (abs(maxrowsum-oldmaxrowsum)/maxrowsum) < 0.001
  then finished := false
  { quit when .1% change } else finished := true;

  oldmaxrowsum := maxrowsum;
end;
until Finished;
Scalar_Mult(Psi,T,Psi,CNPoles);
Matrix_Mult(A,Psi,Phi,CNPoles);
for i:= 1 to CNPoles do
  Phi[i,i] := Phi[i,i] + 1.0;
for i:= 1 to CNPoles do
  Gamma[i] := Psi[i,CNPoles];
{ single input system with B vector: }

```

```

      { B= [ 0 0 0 0 ... 0 1] (transpose))
Plottime := 0.0; PlotIndex := 1; {initialize}
for i:= 1 to CNPoles do Xold[i] := 0.0;
      {init. prev. state}
      Ymax := 0.0; Ymin := 0.0;
Boxuser;
for N := 1 to Nincr do
  begin {begin calculating next state and y}
    {compute input at time Plottime}
    case Inputtype of
      'S' : Uinput := Amplitude;
      'R' : Uinput := Plottime * slope + offset;
      'I' : if plottime = 0 then Uinput := amplitude
            else Uinput := 0.0;
      'W' : Uinput := Amplitude * sin(freq * plottime);
    end; {case}
    Matrix_Vector_Mult(Phi,Xold,XNext,CNpoles);
    {compute new states}
    for i:= 1 to CNpoles do
      Xnext[i] := Xnext[i] + Gamma[i]*Uinput;
    y:= 0.0;
    for i:= 1 to CNPoles do
      begin
        if abs(y) < 1.0E07 then y:= y + C1[i] * Xnext[i]
        else y:= 1.0E07; {max y limit}
      end;
      if SizeZero = CNPoles then
        y := y + NumCoeff[SizeZero+1] * Uinput;
      if y > Ymax then Ymax:= y;
      if y < Ymin then Ymin:= y;
      if N mod 5 = 0 then {plot every 5th point}
      begin
        GraphArray[Plotindex,1] := Plottime;
        GraphArray[Plotindex,2] := y;
        InputArray[Plotindex,1] := Plottime;
        InputArray[Plotindex,2] := Uinput;
        Plotindex := Plotindex + 1;
      end;
      Plottime := Plottime + T; Xold := Xnext;
    end;
    Ymax := 1.1 * Ymax;
  Initgraphic;
  SelectWindow(1);

  drawtext(20,20,1,'O');
  drawtext(20,26,1,'U');
  drawtext(20,32,1,'T');
  drawtext(20,38,1,'P');
  drawtext(20,44,1,'U');
  drawtext(20,50,1,'T');

  drawtext(250,195,1,'TIME(sec)');

  NiceAxes(0,tmax,ymin,ymax,'');
  Selectworld(WorldNdxGlb); SelectWindow(WindowNdxGlb);

  DrawPolygon(GraphArray,1,-(Plotindex-1),0,0,0);
  NiceAxes(0,tmax,ymin,ymax,'');
  SetLineStyle(1); {dashed line for input signal}
  DrawPolygon(InputArray,1,-(Plotindex-1),0,0,0);
  SetLineStyle(0);

```

```
repeat until keypressed;
quit := false;
repeat
  Graph_Menu('Time-Response',DumpGraph,quit);
            {calls print/title menu}
  If DumpGraph then PrintGraphData;
until quit;
LeaveGraphic;

assign(plotfile,'plot.com');
      {re-execution graphics menu program }
execute(plotfile);
end.
```

```

Program Nyquist;
label 1; {label declaration for goto statement}

{$I Typedef.sys}    {graphics routines}
{$I Graphix.sys}
{$I Kernel.sys}
{$I Windows.sys}
{$I Polygon.hgh}
{$I Axis.hgh}

{$I Ut-mod01.inc }   {I/O routines}
{$I Ut-mod02.inc }
{$I Ut-mod03.inc }

{$I GrapMenu.inc}    {graph options menu}
{$I Plotnyqs.inc}    {Nyquist plotting routine}
{$I Boxuser.inc}

type
    ary4  = array [1..11] of real;
    ary6  = array [1..21] of integer;

{$I Polynom.inc}      {Polynomial routine}
{$I Rootfind.inc}     {Polynomial roots finder routine}

var
    Code,I,Count,NumberDecades,
    StartDecade,EndDecade,one      : integer;
    Wf,Wo,Wi,DeltaW,Gain           : Real;
    PlotArray1,PlotArray2,
    MagPhaseArray,FreqArray        : PlotArray;
    ZMagn,ZPhase,PMagn,PPhase,Phase : real;
    TempX,TempY                    : real;
    temp                            : char;
    OpenLoop, PicBig                : boolean;
    j,jj,kk1,m,l,sizezeros,CNpoles : integer;
    Dencoeff,Numcoeff,cdencoeff     : ary4;
    realpartpole,imagpartpole,realpartzero,
    crealpartpole,imagpartzero,cimagpartpole: ary3s;
    PSI                             : ary1s;
    plotfile                        : file;

function Log(X:real):real;
    {computes the base-10 logarithm of X}
begin
    if X=0 then Log:=0 else
        Log := Ln(X)/Ln(10);
end;

function Expon(Y,X:real):real;
    {computes Y raised to X power}
begin
    Expon := exp( X * (ln(Y)));
end;

begin
    Boxuser; one:=1;
    Characteristic equation(A1A,size,Dencoeff);
    for i:=1 to size do
        PSI[i,size]:=B[i,1];
    for jj:=2 to size do
        begin
            for i:=1 to size do
                begin
                    j:=size-jj+1;
                    kk1:=j+1;
                    PSI[i,j]:=Dencoeff[kk1] * B[i,1];
                end
            end
        end
end

```

```

    for l:=1 to size do
    begin
        PSI[i,j]:=PSI[i,j] +A1A[i,l] * PSI[l,kk1];
    end;
end;
for i:=1 to size do
begin
    Numcoeff[i]:=0.0;
    for j:=1 to size do
    begin
        Numcoeff[i]:= Numcoeff[i] + PSI[j,i] * C[1,j];
    end;
end;
for i:=1 to size do
begin
    m:=size+1-i;
    if Numcoeff[m] <> 0.0 then goto 1;
end;
1:sizezeros:=m-1;
Clrscr;
TextColor(lightblue);
writeln('*** Nyquist Plotting Parameters
        ***');TextColor(yellow);
writeln('=====');
PicBig:= false;
Msg('Open (O) or (C) Closed Loop Plot ?',5,5);
repeat
    Input('A','',45,5,2,true,F1,F10);
    temp:=copy(answer,1,1);
    if not(temp in ['O','C']) then Beep(950,350);
until temp in ['O','C'];
if temp = 'O' then OpenLoop := true
    else OpenLoop := false;
Msg('Graph window (B) Big or (S) Select your own
    size?',5,7);
repeat
    input('A','',60,7,2,true,F1,F10);
{sets flag OpenLoop if user selects the open loop}
    temp := copy(answer,1,1);
    if not(temp in ['B','S']) then beep(350,150);
until temp in ['B','S'];
if (temp = 'B') then PicBig := true
    else PicBig := false;

if not (picbig) then
begin
    Msg('Input your first frequency to be
        plotted?',5,9);
    Msg('(example: .01, 1, 100, etc.)',10,10);
    Input('N','',50,9,8,true,F1,F10);
    Val(answer,Wo,code);{Wo is the first plotted freq}

    Msg('Input number of decades do you want
        plotted?',5,12);

    Input('N','',50,12,2,true,F1,F10);
    Val(answer,NumberDecades,code);
end
else
begin
    wo:=0.001;
    NumberDecades:=8;
end;

```

```

root_finder(sizezeros, Numcoeff, realpartzero,
             imagpartzero, one);
root_finder(size, Dencoeff, realpartpole,
             imagpartpole, one);

gain:=Numcoeff[sizezeros+1];
for i:=1 to sizezeros+1 do
begin
  Numcoeff[i]:= Numcoeff[i]/gain;
end;
clrscr; Boxuser;

for i:=1 to maxorder do CDenCoeff[i] := 0.0;
  {initialize}
for i:=1 to SizeZeros + 1 do
  CDenCoeff[i] := Numcoeff[i] * gain;
  {C.L. denominator equals the sum of open loop
   denominator and O.L numerator}
for i:=1 to Size + 1 do
  CDenCoeff[i] := CDenCoeff[i] + Dencoeff[i];
if Size > SizeZeros then Cnpoles:=Size
  else CNPOLES:=SizeZeros;
  {compute new denominator roots}
root_finder(Cnpoles, CDenCoeff, CRealPartPole,
            CImagPartPole, one);

StartDecade := trunc(Log(Wo));
  {compute linear scale to plot }
EndDecade := StartDecade + NumberDecades;
  {log numbers. Also figure step}
Wf := Wo * Expon(10.0, NumberDecades);
DeltaW := Expon((Wf/Wo), 0.0125);
Wi := Wo;

for Count := 1 to 81 do
  {do 81 iterations...arbitrary #}
begin
  if OpenLoop then
  {compute bode numbers if openloop and later if
   closed loop }
  begin
    ZMagn:=1.0; ZPhase:=0.0;
    PMagn:=1.0; PPhase:=0.0; {initialize}
    for i := 1 to SizeZeros do
    {compute magn and phase of zeros for freq step}
    begin
      ZMagn:=ZMagn * Sqrt(Sqr(RealPartZero[i])+
                          Sqr(Wi-ImagPartZero[i]));
      if RealPartZero[i] = 0.0 then
        ZPhase:=ZPhase+pi/2.0 else
      begin
        if realpartzero[i] > 0.0 then
          ZPhase:= ZPhase - pi + ArcTan((Wi-
            ImagPartZero[i])/(-RealPartZero[i]));
        else
          ZPhase:=ZPhase+ ArcTan((Wi-
            ImagPartZero[i])/(-RealPartZero[i]));
        end;
      end;
    end;
    for i := 1 to Size do
      {compute magn and phase of poles for freq step}

```

```

begin
  PMagn:=PMagn * Sqrt(Sqr(RealPartPole[I]) +
    Sqr(Wi-ImagPartPole[I]));
  if RealPartPole[i] = 0.0 then
    PPhase:= PPhase+pi/2.0 else
    begin
      if RealPartPole[i] > 0.0 then
        PPhase:=PPhase - pi + ArcTan((Wi-
          ImagPartPole[i])/(-RealPartPole[I]))
      else
        PPhase:= PPhase+ArcTan((Wi-
          ImagPartPole[i])/(-RealPartPole[i]));
    end;
end;

Phase := Frac((ZPhase - PPhase)/(2*pi)) * (2*pi);

TempX := abs((Gain*ZMagn/PMagn)*cos(Phase));
TempY := abs((Gain*ZMagn/PMagn)*sin(Phase));
if (PicBig) and (TempX > 100) then TempX := 100;
if (PicBig) and (TempY > 100) then TempY := 100;

If Phase<0 then Phase:= Phase+(2*pi);
If ((Phase>(pi/2)) and (Phase<(3*pi/2))) then
  TempX:= -TempX;
If ((Phase>pi) and (Phase<(2*pi))) then
  TempY := -TempY;
MagPhaseArray[count,2] := Phase;
MagPhaseArray[count,1] := Gain*ZMagn/PMagn;
PlotArray1[Count,1] := TempX;
PlotArray2[count,1] := TempX;

PlotArray1[Count,2] := TempY;
PlotArray2[count,2] := -TempY;

FreqArray[Count,1] := wi;

Wi := Wi * DeltaW;          {increment freq step}
end

else
  {perform same steps as above if closed loop requested}
begin
  ZMagn:=1.0;ZPhase:=0.0;PMagn:=1.0;PPhase:=0.0;
  for i := 1 to SizeZeros do
    begin
      ZMagn:=ZMagn * Sqrt(Sqr(RealPartZero[I])+
        Sqr(Wi-ImagPartZero[I]));
      if RealPartZero[I] = 0.0 then
        ZPhase:=ZPhase+pi/2.0 else
        begin
          if RealPartZero[I] > 0.0 then
            ZPhase:=ZPhase - pi + ArcTan((Wi-
              ImagPartZero[i])/(-RealPartZero[i]))
          else
            ZPhase:=ZPhase+ArcTan((Wi-
              ImagPartZero[i])/(-RealPartZero[i]));
        end;
    end;
  end;

  for i := 1 to CNpoles do
    begin
      PMagn:=PMagn * Sqrt(Sqr(CRealPartPole[I])+
        Sqr(Wi-CImagPartPole[I]));
      if CRealPartPole[I] = 0.0 then
        PPhase:=PPhase+pi/2.0 else

```

```

begin
  if CRealPartPole[I] > 0.0 then
    PPhase:=PPhase - pi + ArcTan((Wi-
CImagPartPole[i])/(-CRealPartPole[i]))
  else
    PPhase:=PPhase+ArcTan((Wi-
CImagPartPole[i])/(-CRealPartPole[i]));
  end;
end;

Phase := Frac((ZPhase - PPhase)/(2*pi)) * (2*pi);

TempX := abs({Phase "modulo" 2Pi}
  ((Gain*ZMagn/PMagn)*cos(Phase)));
TempY := abs((Gain*ZMagn/PMagn)*sin(Phase));
if (PicBig) and (TempX > 100) then TempX := 100;
if (PicBig) and (TempY > 100) then TempY := 100;

If Phase<0 then Phase:= Phase+(2*pi);
If ((Phase>(pi/2)) and (Phase<(3*pi/2))) then
  TempX:= -TempY;
If ((Phase>pi) and (Phase<2*pi)) then
  TempY := -TempY;

PlotArray1[Count,1] := TempX;
Plotarray2[count,1] := TempX;

PlotArray1[Count,2] := TempY;
PlotArray2[count,2] := -1.0 * TempY;

Wi := Wi * DeltaW;
end;
Plot_Nyquist(StartDecade,EndDecade,
  NumberDecades,FreqArray,PlotArray1,
  Plotarray2,MagPhaseArray,PicBig,OpenLoop) ;

Assign(Plotfile,'Plot.COM');
Execute(plotfile);
end. {Nyquist}

```

```

Program Root Locus;
{ This program plots the root locus of the plant }
label 1;{label declarations for the goto statement}

{$I Typedef.sys}    {graphics routines}
{$I Graphix.sys}
{$I Kernel.sys}
{$I Windows.sys}
{$I Polygon.hgh}
{$I Axis.hgh}

{$I Ut-mod01.inc}    {I/O routines}
{$I Ut-mod02.inc}
{$I Ut-mod03.inc}

{$I GrapMenu.inc}    {graph options menu}

type
  ary4  = array [1..11] of real;
  ary6  = array [1..21] of integer;
var
  I,J,code,LineCount           : integer;
  PlotPole,PlotZero            : PlotArray;

  PlotRealPole,PlotImagPole    : ary3s;
  DeltaGain,StartGain,EndGain,
  Variable Gain,Xmin,Xmax,Ymin,Ymax,gain: Real;
  Neg_Feedback,dumpgraph,quit  : Boolean;

  list                          : text;
  jj,kk1,m,l,sizezeros,one     : integer;
  holdpoly,dencoeff,numcoeff    : ary4;
  realpartpole,imagpartpole,realpartzero,
  imagpartzero                  : ary3s;
  PSI                           : ary1s;
  plotfile                      : file;

{$I Polynom.inc}
{$I Rootfind.inc}

Procedure PrintGraphData;
  {dumps root locus data to printer}
Begin
  LeaveGraphic; Clrscr;
  repeat
    Textcolor(white);
    gotoxy(20,10);
    writeln(' *** PROGRAM OUTPUT OPTIONS *** ');
    gotoxy(20,13);
    writeln('<P> Printer output ');
    Textcolor(yellow); gotoxy(20,14);
    writeln(' Check Your Printer! ');
    Textcolor(white); gotoxy(20,15);
    writeln('<F> List to File name ');
    gotoxy(20,16);
    writeln(' on the current drive ');
    gotoxy(20,17);
    writeln('<Q> Quit ');
    gotoxy(42,15);textcolor(yellow);
    write('RLOCI.RES'); gotoxy(28,17);
    read(kbd,ch);
    If (ch = 'F') or (ch = 'f') or (ch = 'P') or
      (ch = 'p') then
      begin
        if (ch = 'F') or (ch = 'f') then
          begin

```

```

    gotoxy(24,15);textcolor(red);
    write('PRINTING...');
    assign(list,'Rloci.RES');
    rewrite(list);
end
else
begin
    gotoxy(24,13);textcolor(red);
    write('PRINTING...');
    assign(list,'LST:');
    rewrite(list);
end;

LineCount := 0;
writeln(list);
writeln(list,' ZEROS ');
writeln(list);
write(list,' ');
writeln(list,' REAL
IMAGINARY');
writeln(list); LineCount := LineCount + 6;
for i := 1 to sizeZeros do
begin
    writeln(list,' ',RealPartZero[i]:10:3,
',ImagPartZero[i]:10:3);

    LineCount := LineCount + 1;
end;
writeln(list); writeln(list);
writeln(list,' POLES');
writeln(list);
write(list,' GAIN ');
writeln(list,' REAL
IMAGINARY');
writeln(list); LineCount := LineCount + 7;
Variable_Gain := StartGain;

{compute root locations for varying values of
gain and print them}

DeltaGain := (EndGain-StartGain)/50;
For J:= 1 to 50 do
Begin
    HoldPoly := dencoeff;

    If Neg Feedback then
    For I:= 1 to sizeZeros+1 do
        HoldPoly[I] := HoldPoly[I] +
            (gain*Variable_Gain * numcoeff[I])
    else
    For I:= 1 to sizeZeros +1 do
        HoldPoly[I] := HoldPoly[I] +( gain *
            Variable_Gain * numcoeff[I]);

    Root_Finder(size,HoldPoly,PlotRealPole,
                PlotImagPole,one);
    writeln(list,Variable Gain:10:4);
    LineCount := LineCount + 1;
    for i := 1 to size do
    begin
        writeln(list,' ',i:2,' ',
            PlotRealPole[i]:10:3,
            ',PlotImagPole[i]:10:3);
        LineCount := LineCount + 1;
    end;
    writeln(list);LineCount := LineCount + 1;
    if LineCount > 50 then

```

```

        begin
            writeln(list, chr(12));
            LineCount := 0;
        end;
        variable_gain:=variable_gain+deltaGain;
    end;
end;
until ch in ['Q','q'];
EnterGraphic;
swapscreen;
close(list);
end;

```

```

Begin
    One:=1;{Root Locus Input handler driver}
    P[1]:= '5506N01001-010101';
    P[2]:= '5508N01002-010103';
    P[3]:= '1512N00503-010101';
    P[4]:= '1513N00504-010103';
    P[5]:= '1514N00505-010101';
    P[6]:= '1515N00506-010103';
    P[7]:= '4517A00207T010101';

    Clrscr; TextColor(lightblue);
    writeln('*** ROOT LOCUS PLOTTING PARAMETERS ***');
    TextColor(Yellow);
    writeln('=====');
    writeln;writeln;
    TextColor(green);
    writeln('Input STARTING value for the variable
            gain:');
    writeln;
    writeln('Input ENDING value for the variable
            gain:');
    TextColor(yellow);
    writeln; writeln;

    writeln; TextColor(green);
    writeln('X-Minimum: ');
    writeln('X-Maximum: ');
    writeln('Y-Minimum: ');
    writeln('Y-Maximum: '); writeln;
    writeln('Positive or Negative Feedback? (P or N):');

    Input_Handler('N0107',Escape);
                        {prompts for NEW inputs}
    writeln;writeln;
    writeln('Any changes to these parameters?
            {Y or N}:');
    Input('A','',45,19,2,true,F1,F10);
    If answer='Y' then Input_Handler('C0107',Escape);
                        {prompts for changes}

    Val(filvar[1],StartGain,code);
                        {converts input strings into numeric values}
    Val(filvar[2],EndGain,code);
    Val(filvar[3],Xmin,code);
    Val(filvar[4],Xmax,code);
    Val(filvar[5],Ymin,code);
    Val(filvar[6],Ymax,code);

    If copy(filvar[7],1,1) <> 'N' then
        Neg_feedback := false
    else Neg_feedback := true;

    INITGRAPHIC;{define values for graphics routine}
    NICEAXES(xmin,xmax,ymin,ymax,'');

```

```

Characteristic equation(A1A,size,dencoeff);
{calculate dencoeff from the plant matrices}
for i:=1 to size do
  {calculate numcoeff from the given plant matrices}
  begin
    PSI[i,size]:=B[i,1];
  end;
  for jj:=2 to size do
    begin
      for i:=1 to size do
        begin
          j:=size-jj+1;
          kk1:=j+1;
          PSI[i,j]:=dencoeff[kk1] * B[i,1];
          for l:=1 to size do
            begin
              PSI[l,j]:=PSI[l,j] +A1A[i,l] * PSI[l,kk1];
            end;
          end;
        end;
      end;
    end;
  for i:=1 to size do
    begin
      numcoeff[i]:=0.0;
      for j:=1 to size do
        begin
          numcoeff[i]:=numcoeff[i] + PSI[j,i] * C[1,j];
        end;
      end;
    end;
  for i:=1 to size do
    begin
      m:=size+1-i;
      if numcoeff[m] <> 0.0 then goto 1;
    end;
  1:sizezeros:=m-1;
  {calculate the zeros of the system}
  root_finder(sizezeros,numcoeff,realpartzero,
    imagpartzero,one);

  gain:=numcoeff[sizezeros+1];
  {Convert highest degree numerator coefficient into 1.}
  for i:=1 to sizezeros+1 do
    numcoeff[i]:=numcoeff[i]/gain;

  For I:=1 to sizeZeros do
  Begin
    PlotZero[I,1] := RealPartZero[I];
    PlotZero[I,2] := ImagPartZero[I];
  end; {for}
  Case sizeZeros of
    0 : ;
    1 : begin
      PlotZero[2,1]:=PlotZero[1,1];
      PlotZero[2,2]:=PlotZero[1,2];
      PlotZero[3,1]:=PlotZero[1,1];
      PlotZero[3,2]:=PlotZero[1,2];
      NICEAXES(xmin,xmax,ymin,ymax,'');
      DrawPolygon(PlotZero,1,-3,-3,3,0);
    end;
    2: begin
      PlotZero[3,1]:=PlotZero[1,1];
      PlotZero[3,2]:=PlotZero[1,2];
      NICEAXES(xmin,xmax,ymin,ymax,'');
      DrawPolygon(PlotZero,1,-3,-3,3,0);
    end;
    else
      NICEAXES(xmin,xmax,ymin,ymax,'');
      DrawPolygon(PlotZero,1,sizeZeros,-3,3,0);

```

```

end; {case}
Variable Gain := StartGain;
DeltaGain := (EndGain-StartGain)/50;
           {divide gain to plot 50 points}

For J:= 1 to 50 do
  {calculate and plot 50 points per graph}
Begin
  HoldPoly := dencoeff;
  If Neg_Feedback then
    For I:= 1 to SizeZeros+1 do
      HoldPoly[I] := HoldPoly[I] +
        (gain*Variable_Gain * numcoeff[I])
    else
      For I:= 1 to SizeZeros +1 do
        HoldPoly[I] := HoldPoly[I] +
          (Gain *Variable_Gain * numcoeff[I]);
      Root_Finder(size, HoldPoly, PlotRealPole,
        PlotImagPole, one);
    For I:=1 to size do
      {fill plotting matrix with poles}
    Begin
      PlotPole[I,1] := PlotRealPole[I];
      PlotPole[I,2] := PlotImagPole[I];
    end;
    AxisGlb := true;
    Case size of
      {artificially fill plotting array if fewer
       than 3 points}
      0 : ;
      1 : begin
          PlotPole[2,1] := PlotPole[1,1];
          PlotPole[2,2] := PlotPole[1,2];
          PlotPole[3,1] := PlotPole[1,1];
          PlotPole[3,2] := PlotPole[1,2];
          NICEAXES(xmin,xmax,ymin,ymax,'');
          DrawPolygon(PlotPole,1,-3,-1,3,0);
        end;
      2: begin
          PlotPole[3,1] := PlotPole[1,1];
          PlotPole[3,2] := PlotPole[1,2];
          NICEAXES(xmin,xmax,ymin,ymax,'');
          DrawPolygon(PlotPole,1,-3,-1,3,0);
        end;
      else
          NICEAXES(xmin,xmax,ymin,ymax,'');
          DrawPolygon(PlotPole,1,size,-1,3,0);
        end; {case}
    variable_gain:=variable_gain + Deltagain;
  end;

Repeat until KeyPressed;
quit := false;
repeat
  Graph_Menu('Root Locus',DumpGraph,quit);
  {calls print/title menu}
  If DumpGraph then PrintGraphData;
until quit;
LeaveGraphic;
Assign(plotfile,'plot.com'); {re-execute plot.com}
Execute(plotfile);
end. {root_locus}

```

```

Program Roots;
label 1; {label decleration for goto statement}
{$I Typedef.sys}

TYPE
    ary4 = array [1..11] of real;
    ary6 = array [1..21] of integer;
var
    posCounter,i,j,jj,kk1,m,l,
    cnpoles,sizezeros,one: integer;
    dencoeff,numcoeff,cdencoeff: ary4;
    realpartzero,imagpartzero,
    crealpartpole,cimagpartpole : ary3s;
    PSI
        : ary1s;
    plotfile :file;
    gain      :real;

{$I Polynom.inc}
{$I Rootfind.inc}

BEGIN
    ClrScr; one:=1;
    Characteristic equation(A1A,size,Dencoeff);
    for i:=1 to size do
        PSI[i,size]:=B[i,1];

    for jj:=2 to size do
    begin
        for i:=1 to size do
        begin
            j:=size-jj+1;
            kk1:=j+1;
            PSI[i,j]:=Dencoeff[kk1] * B[i,1];
            for l:=1 to size do
            begin
                PSI[i,j]:=PSI[i,j] +A1A[i,l] * PSI[l,kk1];
            end;
        end;
    end;
    for i:=1 to size do
    begin
        Numcoeff[i]:=0.0;
        for j:=1 to size do
        begin
            Numcoeff[i]:=Numcoeff[i] + PSI[j,i] * C[1,j];
        end;
    end;
    for i:=1 to size do
    begin
        m:=size+1-i;
        if numcoeff[m] <> 0.0 then goto 1;
    end;
    1:sizezeros:=m-1;

    root_finder(sizezeros,Numcoeff,realpartzero,
                imagpartzero,one);

    gain:= Numcoeff[sizezeros+1];
    for i:=1 to sizezeros+1 do
    begin
        Numcoeff[i]:= Numcoeff[i]/gain;
    end;

    for i:=1 to maxorder do
        CDenCoeff[i] := 0.0; {initialize}
    for i:=1 to SizeZeros + 1 do
        CDenCoeff[i] := Numcoeff[i] * gain;

```

```

for i:=1 to Size + 1 do
  CDenCoeff[i] := CDenCoeff[i] + Dencoeff[i];
if Size > SizeZeros then CNPOLES:=Size
{NPoles should always be greater}
else CNPOLES:=SizeZeros;

      {compute new denominator roots}
root_finder(Cnpoles,CDenCoeff,CRealPartPole,
            CImagPartPole,one);

ClrScr; textcolor(lightblue);{on-screen titles}
writeln('      *** Plant Characteristic Equation
      Roots ***');
writeln;writeln;highvideo;
writeln('ROOTS OF THE NUMERATOR: ');

For I:=1 to SizeZeros do {position for output}
begin
  PosCounter := (I mod 2) ;
  If PosCounter = 1 then writeln;

  LowVideo; {write zeros }
  write('s[',I,'] = ',RealPartZero[I]:10:3,
    '+j',ImagPartZero[I]:10:3);
  write(' ');
end;

writeln;writeln; HighVideo;
writeln('ROOTS OF THE DENOMINATOR: ');

For I:=1 to CNPoles do
begin
  {compute on-screen position}
  PosCounter := (I mod 2) ;
  If PosCounter = 1 then writeln;

  LowVideo;
  write('s[',I,'] = ',CRealPARTPOLE[I]:10:3,
    '+j',CImagPARTPOLE[I]:10:3);
  write(' ');
end;{for}

HighVideo;gotoxy(1,24);
write('Press any key to continue or [Shift] [PrtSc]
      for hardcopy.');
```

{check keyboard buffer for value change.
 If number changes by 1 or 2 indicates that shift key
 depressed. If so, then remove "Press any key..."
 prompt from screen so it won't print to printer}

```

keyold := mem[0000:1047]; not_erased := true;
Repeat
  key := mem[0000:1047];
  if ((key=keyold + 1) or (key=keyold + 2)) and
    (not_erased) then
    begin
      GotoXY(1,24); write(' ':80);
      not_erased := false;
    end;
Until KeyPressed;

Assign(plotfile,'Plot.com');
Execute(plotfile);

end. {Program Roots}

```

```

Program Matrix_Manipulation(input,output);

{$I Typedef.SYS}
{$I Box.INC}
{$I Ut-mod01.INC}

var
    help3file,inversefile,determinfile,
    Polynomfile,eigenfile           :file;

Procedure Matrix_mathematics_menu;

var
    i,Tab           : Integer;
    Okchoice        : Set Of Char ;
    Finished        : Boolean;

Procedure MenuItem(Pick:Char;Description :Str80;
                  Color:Integer);

Begin
    TextColor(color);
    Write (',':tab,('{'); TextColor(white); Write(pick);
    TextColor(color); Writeln(')',Description);
End;

Begin
    ClrScr; TextColor(white); Finished := False;
    GoToXY(19,4);
    Write('*** MATRIX MATHEMATICS MENU ***');
    for i := 1 to 4 do writeln(' ');
    Tab := 16;
    MenuItem('D','Determinant of A Matrix',yellow);
    MenuItem('C','Characteristic polynomial of A
                Matrix',yellow);
    MenuItem('I','Inverse of A Matrix',yellow);
    MenuItem('E','Eigenvalues of A Matrix',yellow);
    Writeln;
    Writeln;
    MenuItem('H','Help',lightmagenta);
    MenuItem('Q','Quit to SVS Main menu ',lightmagenta);
    TextColor(green);
    Box; Writeln;
    TextColor(white); GoToXY (40,22);
    Repeat
        Option;
    until ch in ['D','C','I','E','H','Q'];
    case ch of
        'D': begin
            Assign(determinfile,'determin.chn');
            Chain(determinfile);
        end;
        'I': begin
            Assign(inversefile,'inverse.chn');
            Chain(inversefile);
        end;
        'C': begin
            Assign(Polynomfile,'Polynom.chn');
            Chain(Polynomfile);
        end;
        'E': begin
            Assign(eigenfile,'eigen.chn');
            Chain(eigenfile);
        end;
        'H': begin
            Assign(help3File,'help3.CHN');
            Chain(help3File);
        end;
    end;

```

```
      'Q': begin
            Assign(SvsFile, 'svs.COM');
            Execute(SvsFile);
        end;
    End; {case}
End;

Begin {main program}
    Clrscr;
    Repeat
        begin
            Matrix_Mathematics_Menu;
        end;
    Until Finished = True;
End.{main program}
```

```

Program Matrix Determinant(input,output);
label 10;label 20;label 30;label 40;label 50;label 60;
label 70;label 80;

```

```

{$I Typedef.SYS}
{$I Ut-mod01.INC}
{$I Boxuser.INC}

```

```

var
  matrixfile :file;
  list       :text;
  counter,i,j,ii,k,m,n,even      :integer;
  temp_value,value,det,det_correction,
  determinant_old,determinant     :real;
  A1                                     :array1s;

```

```

Begin
  counter := 0;
  Boxuser;
  for i:=1 to size do
  Begin
    for j:=1 to size do
    Begin
      A1[i,j]:= A1A[i,j];
    End;
  End;
  for i:= 1 to size do
  Begin
    k:=i;
    30:if A1[k,i] <> 0.0 then goto 10;
    k:=k+1;
    if (k-size) <= 0.0 then goto 30;
    goto 40;
    10:if (i-k) > 0.0 then goto 40;
    if (i-k) = 0.0 then goto 70;
    for m:=1 to size do
    Begin
      temp_value:=A1[i,m];
      A1[i,m]:=A1[k,m];
      A1[k,m]:=temp_value;
    End;
    counter:= counter+1;
    70:ii:=i+1;
    if ii > size then goto 20;
    for m:=ii to size do
    Begin
      if A1[m,i] = 0.0 then goto 80;
      value:=A1[m,i] / A1[i,i];
      for n:= 1 to size do
      Begin
        A1[m,n]:= A1[m,n] - A1[i,n] * value;
      End;
    End;
    80:End;
  20:End;
  det:=1.0;
  for i:=1 to size do
  Begin
    det:=det * A1[i,i];
  End;
  det_correction:= exp( counter * LN(1));
  determinant_old:=det_correction * det;
  even:= counter mod 2;
  if even <> 0 then goto 60;
  determinant := determinant_old;
  goto 50;
  60:determinant:=-determinant_old;
  goto 50;
  40:determinant:=0.0;
  50:clrscr;writeln;TextColor(yellow);

```

```

Writeln('The given matrix is : '); writeln;
for i:= 1 to size do
Begin
  for j:= 1 to size do
  Begin
    Write(' ',A1A[i,j]:11);
  End;Writeln;
End;Writeln;Writeln;
Write('The determinant value is :');
TextColor(white);writeln(determinant);

keyread(key,keyold,not_erased);

Repeat
  box printer;gotoxy(58,15);textcolor(yellow);
  writeln('DETERMIN.RES');
  gotoxy(1,25);write(' ');
  gotoxy(49,17);
  read(kbd,ch);
  if (ch='F') or (ch='f') or (ch='P') or (ch='p')
  then begin
    if (ch='F') or (ch='f') then
    begin
      gotoxy(45,15);textcolor(red);
      write('PRINTING.....');
      Assign(list,'Determin.RES');
      Rewrite(list);
    end;
    if (ch='P') or (ch='p') then
    begin
      gotoxy(45,13);textcolor(red);
      write('PRINTING.....');
      Assign(list,'LST:');
      Rewrite(list);
    end;
    writeln(list,'DETERMINANT
    RESULT ');
    writeln(list);
    writeln(list,'The given matrix is:');
    writeln(list);
    for i:=1 to size do
    begin
      for j:=1 to size do
      begin
        write(list,' ',A1A[i,j]:11);
      end;writeln(list);
    end;writeln(list);writeln(list);
    Write(list,'The determinant value is : ');
    writeln(list,determinant);
    writeln(list);writeln(list);
    close(list);
  end;
Until ch in ['Q','q'];

Assign(matrixfile,'matrix.com');
Execute(matrixfile);

End.

```

```

Program Matrix inverse(input,output);
{$I Typedef.SYS}
{$I Ut-mod01.INC}
{$I Boxuser.inc}
type
    ary4    = array [1..11] of real;
    ary6    = array [1..21] of integer;
var
    matrixfile :file;
    list       :text;
    polynom_coeff :ary4;
    Ainverse    :ary1s;
    singular,step,stepper :integer;
{$I Polynom.inc}
{$I Inverse.inc}
begin
    clrscr; Boxuser; singular:=0;
    characteristic_equation(A1A,size,Polynom_coeff);
inverse_find_matrix(A1A,polynom_coeff,size,Ainverse,
                    polynom_coeff,singular);
    clrscr;textcolor(yellow);
    writeln('The given matrix is: '); writeln;
    for step:=1 to size do
    begin
        for stepper:=1 to size do
        begin
            write(' ',A1A[step,stepper]:11);
            end;writeln;
        end;writeln;
        if singular=1 then
            writeln('The matrix is singular.')
        else
        begin
            writeln('The inverted matrix is:');writeln;
            for step:=1 to size do
            begin
                for stepper:=1 to size do
                begin
                    write(' ',Ainverse[step,stepper]:10,' ');
                    end;writeln;
                end;
            end;
        end;
        keyread(key,keyold,not_erased);
    Repeat
        box printer;gotoxy(58,15);textcolor(yellow);
        writeln('"INVERSE.RES"'); gotoxy(1,25););
        write(' ');
        gotoxy(49,17);
        read(kbd,ch);
        if (ch='F') or (ch='f') or (ch='P') or (ch='p')
        then begin
            if (ch='F') or (ch='f') then
            begin
                gotoxy(45,15);textcolor(red);
                write('PRINTING.....RES' );
                Assign(list,'Inverse.RES');
                Rewrite(list);
            end;
            if (ch='P') or (ch='p') then
            begin

```

```
        gotoxy(45,13);textcolor(red);
        write('PRINTING:');... ');
        Assign(list,'LST:');
        Rewrite(list);
    end;
    writeln(list,'          MATRIX INVERSE
    writeln(list); RESULT ');
    writeln(list,'The given matrix is:');
    writeln(list);
    for step:=1 to size do
    begin
        for stepper:=1 to size do
        begin
            write(list,' ',A1A[step,stepper]:11);
            end;writeln(list);
        end;writeln(list);writeln(list);
    begin
        writeln(list,'The inverted matrix is:');
        writeln(list);
        for step:=1 to size do
        begin
            for stepper:=1 to size do
            begin
                write(list,' ',
                Ainverse[step,stepper]:10,' ');
                end;writeln(list);
            end;
        end;
        writeln(list);writeln(list);
        close(list);
    end;
    Until ch in ['Q','q'];
    Assign(matrixfile,'matrix.com');
    Execute(matrixfile);
end.
```

```

Program Matrix Manipulation(input,output);
{$I Typedef.SYS}
{$I Ut-mod01.INC}{$I Boxuser.inc}
type
    ary4    = array [1..11] of real;
    ary5    = array [1..21] of real;
    ary6    = array [1..21] of integer;

var
    matrixfile :file;
    list       :text;
    c2,C1      :ary4;
    RealPart,ImaginaryPart:ary3s;
    i,j,one    :integer;

{$I Polynom.inc}
{$I Rootfind.inc}

begin
    clrscr;Boxuser;one:=1;
    characteristic_equation(A1A,size,C1);

    root_Finder(size,C1,RealPart,ImaginaryPart,one);
    clrscr;Textcolor(yellow);
    writeln('The given matrix is:');writeln;
    for i:=1 to size do
    begin
        for j:=1 to size do
        begin
            write(' ',A1A[i,j]:11);
        end;writeln;
    end;writeln;writeln;
    writeln('The eigenvalues of the matrix are: ');
    writeln('                                REAL PART
                                IMAGINARY PART');
    for i:=1 to size do
    begin
        write('                                ');
        write(RealPart[i]:3);
        write(' + j ');
        write(ImaginaryPart[i]:3);writeln;
    end;

    keyread(key,keyold,not_erased);

    Repeat
        box_printer;gotoxy(58,15);textcolor(yellow);
        writeln('"EIGEN.RES"'); gotoxy(1,25);
        write(' ');
        gotoxy(49,17);
        read(kbd,ch);
        if (ch='F') or (ch='f') or (ch='P') or (ch='p')
        then begin
            if (ch='F') or (ch='f') then
            begin
                gotoxy(45,15);textcolor(red);
                write(' PRINTING...RES ');
                Assign(list,'Eigen.RES');
                Rewrite(list);
            end;
            if (ch='P') or (ch='p') then
            begin
                gotoxy(45,13);textcolor(red);
                write(' PRINTING... ');
                Assign(list,'LST:');
                Rewrite(list);
            end;
        end;
    end;

```

```
writeln(list,' RESULT '); EIGENVALUES
writeln(list);
writeln(list,'The given matrix is:');
writeln(list);
for i:=1 to size do
begin
  for j:=1 to size do
  begin
    write(list,' ',A1A[i,j]:11);
    end;writeln(list);
  end;writeln(list);writeln(list);
  writeln(list,'
The eigenvalues of the matrix are: ');
  writeln(list,' REAL PART');
  for i:=1 to size do
  begin
    write(list,' ');
    write(list,RealPart[i]:10);
    write(list,' + j');
    write(list,ImaginaryPart[i]:10);
    writeln(list);
  end;
  writeln(list);writeln(list);
  close(list);
end;
Until ch in ['F','f','P','p','Q','q'];
Assign(matrixfile,'matrix.com');
Execute(matrixfile);

end.
```

```

Program Matrix Characteristic_Equation(input,output);
{$I Typedef.SYS}
{$I Ut-mod01.INC}
{$I Boxuser.inc}
type
    ary4      = array [1..11] of real;
    ary6      = array [1..21] of integer;

var
    matrixfile :file;
    list       :text;
    C1         :ary4;
    i,j,vertpos,horizpos,poscounter :integer;
    exponent   :string[2];
{$I Polynom.inc}
begin
    clrscr;Boxuser;
    characteristic_equation(A1A,size,C1);clrscr;
    Textcolor(yellow); writeln;
    writeln('The given matrix is : ');writeln;
    for i:=1 to size do
        begin
            for j:=1 to size do
                begin
                    write(' ',A1A[i,j]:11);
                end;writeln;
            end;writeln;writeln;
            writeln('The characteristic polynomial of given
                    matrix is: ');
            vertpos:=size+7;
            for i:=size+1 downto 1 do
                begin
                    j:=size+1-i;
                    poscounter:=(j mod 4) +1;horizpos:=poscounter * 14;
                    if poscounter = 1 then vertpos:=vertpos+2;
                    if i <> 1 then
                        begin
                            gotoXY(horizpos-10,vertpos);write(C1[i]:7:4);
                            msg(' S +',horizpos,vertpos);
                            str(i-1:2,exponent);
                            msg(exponent,horizpos+1,vertpos-1);
                        end
                    else
                        begin
                            gotoXY(horizpos-10,vertpos);write(C1[i]:7:4);
                        end;
                end;
            end;
            keyread(key,keyold,not_erased);
            Repeat
                box printer;gotoxy(58,15);textcolor(yellow);
                writeln('POLYNOM.RES');
                gotoxy(1,25);write(' ');
                gotoxy(49,17);
                read(kbd,ch);
                if (ch='F') or (ch='f') or (ch='P') or (ch='p')
                    then begin
                        if (ch='F') or (ch='f') then
                            begin
                                gotoxy(45,15);textcolor(red);
                                write('PRINTING ');
                                Assign(list,'Polynom.RES');
                                Rewrite(list);
                            end;
                    end;
            until keyold=key;
        end;
    end;
end;

```

```
if (ch='P') or (ch='p') then
begin
  gotoxy(45,13);textcolor(red);
  write('PRINTING.;;... ');
  Assign(list,'LST:');
  Rewrite(list);
end;
writeln(list,' POLYNOMIAL
RESULT ');
writeln(list);
writeln(list,'The given matrix is:');
writeln(list);
for i:=1 to size do
begin
  for j:=1 to size do
  begin
    write(list,' ',A1A[i,j]:11);
  end;writeln(list);
end;writeln(list);writeln(list);
write(list,'The characteristic polynomial of
given matrix is:');
writeln(list,' ( Descending power of S ) ');
writeln(list);
for i:=size+1 downto 1 do
begin
  write(list,C1[i]:7:4,' ');
  str(i-1:2,exponent);
end;
writeln(list);writeln(list);
close(list);
end;
Until ch in ['Q','q'];
Assign(matrixfile,'matrix.com');
Execute(matrixfile);
end.
```

```

Program Control(input,output);
{$I Typedef.SYS}
{Program type and variable definitions}
{$I Ut-mod01.INC} {input utility programs}

type
  Ary1    = Array[1..11,1..11] of real;
var
  Stepping,Steps,step,steppings,
  Temp1,Temp2,value1,value2,m,mplus,
  last_rank,k,value1plus,kplus      :integer;
  new_matrix1,new_matrix2            :ary1s;
  big_matrix                         :ary1;
  list                               :text;

{$I control.inc} {finds controllability result}
{$I Boxuser.inc}
Begin
  value1 := 1; value1plus:= value1;
  ClrScr;Textcolor(yellow);
  Writeln('The A matrix is : ');writeln;
  For steps := 1 to size do
  Begin
    For stepping := 1 to size do
    Begin
      Write(' ',A1A[steps,stepping]:11);
    End; Writeln;
  End;writeln;writeln;
  Writeln('The B Matrix is : ');writeln;
  For steps := 1 to SIZE do
  Begin
    for stepping:=1 to ni do
    begin
      Write(' ',B[steps,stepping]:11);
    end;writeln;
  End;writeln;writeln;
  For steps:=1 to size do
  Begin
    For stepping :=1 to ni do
    Begin
      new_matrix2[steps,stepping] := B[steps,stepping];
      new_matrix1[steps,stepping] := B[steps,stepping];
    End;
  End; mplus:= size-1; value2:=value1;
  For steps := 1 to mplus do
  Begin
    matrix_multiplication(A1A,new_matrix1,
                          new_matrix1,size,size,value2);
    step := value1 +1; value1 := value1 + value2;
    For k := 1 to size do
    Begin
      For stepping := step to value1 do
      Begin
        steppings := stepping-step+1;
        new_matrix2[k,stepping]:=
          new_matrix1[k,steppings];
      End;
    End;
  End;
  For steps :=1 to size do
  Begin
    For stepping:=1 to value1 do
    Begin
      big_matrix[steps,stepping] :=
new_matrix2[steps,stepping];
    End;
  End;

```

```

End; mplus:=1;
For steps :=1 to size do
Begin
  Matrix_reduction(big_matrix,mplus,size,last_rank);
  mplus := last_rank +1;
End;
If last_rank = size then
Writeln('The system is controllable. ');Writeln;
If last_rank < size then
Writeln('The system is uncontrollable. ');Writeln;
value1:=value1plus;
KEYREAD(KEY,KEYOLD,NOT_ERASED);
repeat
  box_printer;gotoxy(58,15);textcolor(yellow);
  writeln('CONTROL.RES');
  gotoxy(1,25);write(' ');
  gotoxy(49,17);
  Read(kbd,ch);
  If (ch='F') or (ch='f') or (ch='p') or (ch='P')
  then Begin
    If (ch='F') or (ch='f') then
      Begin
        gotoxy(45,15);Textcolor(red);
        writeln('PRINTING. .... ');
        Assign(list,'control.RES');
        Rewrite(list);
      End;
    If (ch='P') or (ch='p') then
      Begin
        gotoxy(45,13);Textcolor(red);
        writeln('PRINTING. .... ');
        Assign(list,'LST:');
        Rewrite(list);
      End;
    Writeln(list, '          CONTROLLABILITY
          RESULT ');
    Writeln(list);
    Writeln(list,'The Plant matrix A is : ');
    writeln(list);
    For steps := 1 to size do
      Begin
        For stepping := 1 to size do
          Begin
            Write(list, '          A1A[steps,stepping]:11);
          End;Writeln(list);
        End;Writeln(list);Writeln(list);
        Writeln(list,'The input Matrix B is : ');
        writeln(list);
        For steps := 1 to SIZE do
          Begin
            For stepping := 1 to ni do
              Begin
                Write(list, '          B[steps,stepping]:11);
              End;writeln(list);
            End;Writeln(list);Writeln(list);
            If last_rank = size then
              Writeln(list,'The system is controllable. ');
            Writeln;
            If last_rank < size then
              Writeln(list,'The system is uncontrollable. ');
            Writeln;
            writeln(list);writeln(list);
            Close(list);
          End;
        Until Ch in ['Q','q'];
        Assign(SvsFile,'svs.COM');
        Execute(SvsFile);
      End.

```

```

Program Observability(input,output);
label 1;label 2;label 3;label 4;

{$I Typedef.SYS} {program type and definitions}

type
  Ary1   = Array[1..11,1..11] of real;

var
  stepping,steps,step,temp1,temp2,
  add_no,value2,last_rank      : integer;
  matrix                       : ary1s;
  matrix_matrix                : ary1;
  list                         : text;
{$I Boxuser.inc}
{$I Control.inc}
{$I Ut-mod01.inc} {utility input routines}

Begin
  ClrScr; Textcolor(yellow);
  Writeln('The Plant matrix A is: ');writeln;
  For steps := 1 to size do
  Begin
    For stepping := 1 to size do
    Begin
      Write(' ',A1A[steps,stepping]:11);
    End;Writeln;
  End;Writeln;writeln;
  Writeln('The output Matrix C is : ');writeln;
  For stepping := 1 to no do
  begin
    for steps:=1 to size do
    begin
      Write(' ',C[stepping,steps]:11);
    end;writeln;
  end;writeln;writeln;
  add_no:=no; value2 :=1;
  For steps:= 1 to no do
  Begin
    For stepping:=1 to size do
    Begin
      matrix_matrix[steps,stepping] :=
        C[steps,stepping];
      matrix[steps,stepping] := C[steps,stepping];
    End;
  End;
  4:matrix_reduction(matrix_matrix,add_no,
    size,last_rank);
    {calculate the rank of the matrix}
  If last_rank < size then GoTo 2;
  Writeln('The system is observable with index ',
    value2,'. ');Writeln;
  KEYREAD(KEY,KEYOLD,NOT_ERASED);
  GoTo 1;
  2:If value2 < size then GoTo 3;
  Writeln('The system is unobservable. ');
  KEYREAD(KEY,KEYOLD,NOT_ERASED);
  Writeln;GoTo 1;
  3:value2 := value2+1;
  matrix_multiplication(matrix,A1A,matrix,no,
    size,size);
    {multiply the A and C matrices }
  For steps := 1 to no do
  Begin
    step:= last_rank + steps;
    for stepping:=1 to size do
    Begin

```

```

        matrix_matrix[step,stepping] :=
            matrix[steps,stepping];
    End;
End;
add_no := last_rank + no;
GoTo 4;
1:repeat
    Box printer;gotoxy(58,15);Textcolor(yellow);
    writeln('"OBSER.RES"'); gotoxy(1,25);,);
    write('
    gotoxy(49,17);
    Read(Kbd,ch);
    If (ch='P') or (ch='p') or (ch='F') or (ch='f')
    then Begin
        If (ch='F') or (ch='f') then
            Begin
                GotoXY(45,15);textcolor(red);
                writeln('PRINTING.:RES');,);
                Assign(list,'obser.RES');
                {print observability output }
                Rewrite(list);{to file on the current drive}
            End;
            If (ch='P') or (ch='p') then
                Begin
                    gotoxy(45,13);textcolor(red);
                    writeln('PRINTING.:.....');,);
                    Assign(list,'LST:');
                    {print observability result to the printer}
                    Rewrite(list);
                End;
                Writeln(list,'
                                OBSERVABILITY RESULT ');,);

                Writeln(list);Writeln(list);
                Writeln(list,'The Plant matrix A is :');
                Writeln(list);
                For steps := 1 to size do
                    Begin
                        For stepping := 1 to size do
                            Begin
                                Write(List,' ',A1A[steps,stepping]:11);
                            End;Writeln(list);
                        End;Writeln(list);Writeln(list);
                        Writeln(list,'The Output Matrix C is : ');
                        writeln(list);
                        For steps:= 1 to no do
                            begin
                                for stepping:=1 to size do
                                    begin
                                        Write(list,' ',C[steps,stepping]:11);
                                    end;writeln(list);
                                end;Writeln(List);Writeln(list);
                                If last_rank = size then
                                    Writeln(list,'The system is observable with
                                        index ',value2,',');
                                If (value2 >= size ) and (last_rank < size) then
                                    Writeln (list,'The system is unobservable. ');
                                writeln(list);writeln(list);
                                Close(list);
                            End;
                        Until Ch in ['Q','q' ];
                    Assign(SvsFile,'svs.COM');
                    {re-execute SVS main program}
                    Execute(SvsFile);
                End.

```

```
Program Luenberg_observer(input,output);
{$I Typedef.SYS}-
```

```
{ $I Ut-mod01.INC }
{ $I Ut-mod02.inc }
```

```
label 940;label 960;label 22;label 28; label 950;
```

```
Type
```

```
ary1      = array [1..10] of real;
ary2      = array [1..30,1..30] of real;
ary3      = array [1..30] of real;
ary4      = array [1..11] of real;
ary6      = array [1..11] of integer;
```

```
Var
```

```
rr1,rim,ooo,oooo      :real;
i,NR1,l,mm,nrm,nrp,ixx,jxx,s, one,
nrm,nr,r,rp,k,t,o,umran,ii,jj,
code,vertpos,horizpos,poscounter :integer;
h,a1,e                :ary1;
coeff,coef,desired_feedback      :ary4;
f                            :ary1s;
beta,x                 :ary3;
phi,u                  :ary2;
list                   :text;
strg,exponent          :string[2];
change                 :boolean;
specification          :string[5];
Realpartvalue,imaginarypartvalue :ary3s;
temp,inputtype        :char;
```

```
{ $I Ut-mod03.inc }
```

```
{ $I Luenberg.inc }
```

```
{ $I Pole.inc }
{ $I Rootfind.inc }
{ $I Boxuser.inc }
```

```
Begin
```

```
ClrScr; luen:=true;one:=1;
for i:=1 to size do
  for l:=1 to 1 do
    E[i]:=B[i,l];
gotoxy(1,22);
invvideo('Press <ESC> to change it!',      ');
gotoxy(1,23);
invvideo('Then type your input with <ENTER> key ');
gotoxy(1,1); Textcolor(lightblue);
writeln('*** Luenberger Observer Design
Parameters ***');
TextColor(yellow);
writeln('=====');
TextColor(yellow);
msg('Input degree of observer ( 10 max)',1,5);
Repeat
  input('N',',',40,5,3,true,F1,F10);
  val(answer,r,code);
  if (r > 10) and (r < 1) then beep(900,350);
until (r <= 10) and (r > 0);
rp:=r+1;
msg('Input the Desired Feedback Coefficients in
Factored <F> Form ',1,7);
msg('or Coefficient <C> Form ',1,8);
repeat
  input('A',',',64,7,2,true,F1,F10);
  temp:=copy(answer,1,1);
```

```

    if not (temp in ['F','C']) then beep(900,350);
until temp in ['F','C'];
inputtype:=temp;
Case inputtype of
  'F':begin
    if change then
      for i:=1 to size do
      begin
        str(realpartvalue[i]:8:2,Filvar[2*i+9]);
        str(imaginarypartvalue[i]:8:2,Filvar[2*i+10]);
      end;
      input_Factored('POLES',size,RealPartvalue,
                    ImaginaryPartvalue);
    end;
  'C':begin
    if change then
      for i:=1 to size-1 do
      begin
        str(Desired_feedback[size-1-i]:8:2,
            Filvar[size+22-i]);
      end;
      input_coef('POLES',size-1,Desired_feedback);
    end;
end;ClrScr;Writeln;gotoxy(1,22);
invvideo('Press <ESC> to change it!',
gotoxy(1,23);
invvideo('Then type your input with <ENTER> key ');
gotoxy(1,1); TextColor(lightblue);
Writeln('*** Luenberger Observer Design
        Input Parameters ***');
textcolor(yellow);
writeln('=====');

textcolor(green);
msg('Input observer characteristic polynomial
        Factored (F) Form ',1,5);
msg('        or Coefficient (C) Form ',1,6);
repeat
  input('A','C',65,5,2,true,F1,F10);
  temp:= copy(answer,1,1);
  if not (temp in ['F','C']) then beep(900,350);
until temp in ['F','C'];
if temp = 'C' then goto 950;

if change then
for i:=1 to r do
begin
  str(realpartvalue[i]:8:2,Filvar[2*i+9]);
  str(imaginarypartvalue[i]:8:2,Filvar[2*i+10]);
end;
input_Factored('POLES',r,RealPartvalue,
              ImaginaryPartvalue);

Polynomial_of_roots(r,Realpartvalue,
                  Imaginarypartvalue,coef); GoTo 960;

950:if change then
for i:= 1 to r do
begin
  str(coef[r-i]:8:2,Filvar[r+22-i]);
end;

input_coeff('POLES',r,coef);
root_finder(r,coef,realpartvalue,
            imaginarypartvalue,one);

```

```

ClrScr;Writeln;Writeln;
Writeln('The observer eigenvalues are : ');
Writeln('      REAL PART      IMAGINARY PART');
for i:= 1 to r do
Begin
  Write('      ',realpartvalue[i]:10,'      +      ',
    imaginarypartvalue[i]:10,'j '); Writeln;
End;
Writeln;Writeln;
Writeln('The Observer characteristic polynomial
coefficints in ascending powers of S');
Writeln;
for i:=1 to rp do
Begin
  Write(coef[i]:10,' ');
End; Writeln;Writeln;
960:KEYREAD(KEY,KEYOLD,NOT_ERASED);ClrScr;
for i:= 1 to 30 do
Begin
  for l:= 1 to 30 do
  Begin
    PHI[i,l] :=0.0;
  End;
End;
for i:= 1 to r do
Begin
  H[i]:= 0.0;
End;
H[1]:=1.0;
for i := 1 to r do
Begin
  for l:= 1 to r do
  Begin
    F[i,l]:= 0.0;
  End;
  F[i,1] := -COEF[rp-i];
  if i = r then goto 28;
  F[i,i+1] := 1.0;
28:End;
for i:=1 to 20 do
begin
  X[i]:=0.0;
  BETA[i]:=0.0;
End;
nr := size * r;
for i:= 1 to nr do
Begin
  BETA[i] := 0.0;
End;
nrp := nr+1;
nrn := nr +size;
for i := 1 to size do
Begin
  BETA[nr+i] := desired_feedback[i];
End;
for i := 1 to r do
Begin
  ii:= size*(i-1);
  for l:= 1 to r do
  Begin
    jj:= size * (l-1);
    for k:= 1 to size do
    Begin
      PHI[ii+k,jj+k] := -F[i,l];
    End;
  End;
End;
for ii:= 1 to r do

```

```

Begin
  i:=(ii-1) * size;
  for l:= 1 to size do
  Begin
    for k:= 1 to size do
    Begin
      PHI[i+l,i+k] := PHI[i+l,i+k]+ A1A[k,l];
    End;
  End;
End;
for ii:= 1 to r do
Begin
  i:= size * (ii-1);
  for l:= 1 to size do
  Begin
    PHI[nr+l,i+l] := H[ii];
  End;
End;
for i:= 1 to size do
Begin
  for l:= 1 to no do
  Begin
    PHI[nr+i,nr+l] := C[l,i];
  End;
End;
nrm:= nr+no;
for ii:= 1 to r do
Begin
  i:= size * (ii-1);
  t:= nrm + no *(ii-1);
  for k:= 1 to no do
  Begin
    for s:= 1 to size do
    Begin
      PHI[i+s,t+k] := -C[k,s];
    End;
  End;
End;
mm:= r * (size+no) + no;

Boxuser;
TextColor(white);

linear_equation(PHI,NRN,MM,BETA,X,K,U);

for ii:= 1 to r do
Begin
  i:= size * (ii-1);
  AJ[ii] := 0.0;
  for o:= 1 to size do
  Begin
    AJ[ii]:=AJ[ii] +E[o] *X[i+o];
  End;
End;
nr1:= nr+1;
ClrScr;Writeln;Writeln;textcolor(yellow);
Writeln('The F Matrix is : ');Writeln;
for i:= 1 to r do
Begin
  for o:=1 to r do
  Begin
    Write(' ',F[i,o]:11);
  End;Writeln;
End;
KEYREAD(KEY,KEYOLD,NOT_ERASED); ClrScr;Writeln;
textcolor(yellow);Writeln('The G Matrix is : ');
Writeln;
for ii:= 1 to r do

```

```

Begin
  l:= no *(ii-1) + nrm;
  for o:= 1 to no do
    Begin
      Write(' ',X[o+i]:11);
    End;Writeln;
  End;writeln;writeln;
  Writeln('The G2 Matrix is : ');Writeln;
  for i:= 1 to r do
    Begin
      Writeln(' ',AJ[i]:11);
    End;
  KEYREAD(KEY,KEYOLD,NOT_ERASED);
  ClrScr;textcolor(yellow);Writeln;
  Writeln('The output feedback coefficients are
    ( Ascending powers of S ) :');Writeln;
  for o:= nr1 to nrm do
    Begin
      Write(' ',X[o]:11);
    End;Writeln;writeln;
  Writeln('The compensator feedback coefficients are
    (ascending powers of S ) :');
  writeln;for i:= 1 to r do
    Begin
      Write(' ',H[i]:11);
    End;
  KEYREAD(KEY,KEYOLD,NOT_ERASED);

Repeat
  Box printer;textcolor(yellow);gotoxy(58,15);
  write('"LUENBERG.RES" ');
  gotoxy(1,25);write(' ');
  gotoxy(49,17);
  Read(kbd,ch);
  if (ch='P') or (ch='p') or (ch='F') or (ch='f') then
    Begin
      if (ch='F') or (ch='f') then
        Begin
          gotoxy(45,15);Textcolor(red);
          write('PRINTING..... ');
          Assign(list,'luenberg.RES');
          Rewrite(list);
        End;
      if (ch='P') or (ch='p') then
        Begin
          gotoxy(45,13);textcolor(red);
          write('PRINTING..... ');
          Assign(list,'LST:');
          Rewrite(list);
        End;
      Writeln(list,'LUENBERGER OBSERVER RESULT ');
      For i:=1 to 2 do writeln(list);
      Writeln(list,'The plant matrix A is : ');
      Writeln(list);
      for i:=1 to size do
        Begin
          for l:=1 to size do
            Begin
              Write(list,' ',A1A[i,l]:11);
            End;Writeln(list);
          End;Writeln(list);
          Writeln(list,'The input matrix B is :') ;
          Writeln(list);
          for i:=1 to size do
            Begin
              for l:=1 to 1 do
                Begin
                  Write(list,' ',B[i,l]:11);

```

```

        End;Writeln(list);
End;Writeln(list);
Writeln(list,'The output Matrix C is:');
Writeln(list);
for i:= 1 to no do
Begin
    for l:=1 to size do
        Begin
            Write(list,' ',C[i,l]:11);
            End;Writeln(list);
        End;Writeln(list);
Writeln(list,'The desired feedback coefficients
are : ');Writeln(list);
for i:=1 to size do
Begin
    for l:=1 to 1 do
        Begin
            Write(list,' ',desired_feedback[i]:11);
            End;Writeln(list);
        End;Writeln(list);
Writeln(list,'The observer eigenvalues are : ');
Writeln(list,'REAL PART      IMAGINARY PART');
for i:= 1 to r do
Begin
    Write(list,' ',realpartvalue[i]:10,' +
imaginarypartvalue[i]:10,' j '); Writeln(list);
End;Writeln(list);
Writeln(list,'The Observer characteristic
polynomial coefficints in ascending powers of S');

Writeln(list);
for i:=1 to rp do
Begin
    Write(list,' ',coef[i]:11);
End;Writeln(list);writeln(list);
Writeln(list,'The F Matrix is : ');Writeln(list);
for i:= 1 to r do
Begin
    for o:=1 to r do
        Begin
            Write(list,' ',F[i,o]:11);
            End;Writeln(list);
        End;Writeln(list);
Writeln(list,'The G1 Matrix is : ');Writeln(list);
for ii:= 1 to r do
Begin
    i:= no *(ii-1) + nrm;
    for o:= 1 to no do
        Begin
            Write(list,' ',X[o+i]:11);
            End;Writeln(list);
        End;Writeln(list);
Writeln (list,'The G2 Matrix is : ');Writeln(list);

for i := 1 to r do
Begin
    Writeln(list,' ',AJ[i]:11);
End;Writeln(list);
Writeln (list,'The output feedback coefficients
are : ');Writeln(list);
for o:= nr1 to nrm do
Begin
    Write(list,' ',X[o]:11);
End;Writeln(list);writeln(list);
Writeln(list,'The compensator feedback
coefficients are : ');Writeln(list);
for i:= 1 to r do
Begin

```

```
        Write(list,' ',H[i]:11);
    End;
    for i:=1 to 3 do writeln(list);
    Close(list);
End;
Until ch in ['Q','q'];
Assign(SvsFile,'svs.COM');
Execute(SvsFile);
End.
```

```

program optimal control(input,output);
{$I Typedef.sys}
{$I Graphix.sys}
{$I Kernel.sys}
{$I Windows.sys}
{$I Polygon.hgh}
{$I Axis.hgh}

{$I Ut-mod01.INC}      { I/O procedures}
{$I Ut-mod02.INC}
{$I Ut-mod03.inc}

{$I Grapmenu.inc}
{$I Boxuser.inc}
label 2;label 3;
const
  xarray : string[14] = 'TIME INTERVALS';
var
  GTNY   : array [1..80] of real;
  GTN    : array [1..10,1..80] of real;
  PSI,P1,D3,D4,FI,Q,gamma : array1s;
  D2,D1,GT : array3s;
  ti,i,j,l,nterm,kk,code : integer;
  y,ymax,ymin,r,denominator,negatif,si,result : real;
  grapharray,grapharray1 : plotarray;
  dumpgraph,quit : boolean;
  list : text;
  ans : char;

{$I Optimal.inc}

Procedure PrintGraphData;
  {this procedure dumps optimal result data to printer}

Begin
  LeaveGraphic;Clrscr; Textcolor(yellow);
  Center('*** PROGRAM OUTPUT OPTIONS ***',1,10,80);
  TextColor(green);
  msg('Press <P> Print results to the printer ',1,12);
  msg('Press <F> List results to file name',1,13);
  msg(' "OPTIMAL.RES" on the current drive',1,13);
  msg('Press <Q> Quit ',1,14);
  repeat
    Read(kbd,ch);
    If (ch = 'F') or (ch = 'f') or (ch = 'p')
      or (ch = 'P') then
      begin
        if (ch = 'F') or (ch = 'f') then
          begin
            Assign(list,'Optimal.RES');
            Rewrite(list);
          end
        else
          begin
            assign(list,'LST:');
            rewrite(list);
          end;
        Writeln(list);
        write(list);
        writeln(list,'OPTIMAL CONTROL RESULT ');
        writeln(list);
        write(list,'The order of the system is:');
        writeln(list,size:2);
        write(list,'The number of time intervals is:');
        writeln(list,si:2);
        write(list,'The scaler R is:');
        writeln(list,r:7:4);
      end;
  until ch = 'Q';

```

```

write(list,'The sample interval is:');
writeln(list,ti:4);
writeln(list);
writeln(list,'The A matrix is:');writeln(list);
for l:= 1 to size do
begin
  for j:= 1 to size do
  begin
    write(list,' ',A1A[l,j]:11);
    end;writeln(list);
  end; writeln(list);
  writeln(list,'The B matrix is:');writeln(list);
  for l:= 1 to size do
  begin
    for j:= 1 to ni do
    begin
      write(list,' ',B[l,j]:11);
      end;writeln(list);
    end; writeln(list);
    writeln(list,'The Q matrix is:');writeln(list);
    for l:= 1 to size do
    begin
      for j:= 1 to size do
      begin
        write(list,' ',Q[l,j]:11);
        end;writeln(list);
      end; writeln(list);
      writeln(list,'The FI matrix is:');writeln(list);
      for l:= 1 to size do
      begin
        for j:= 1 to size do
        begin
          write(list,' ',FI[l,j]:11);
          end;writeln(list);
        end; writeln(list);
        writeln(list,'The GAMMA matrix is:');
        writeln(list);
        for l:= 1 to size do
        begin
          for j:= 1 to ni do
          begin
            write(list,' ',GAMMA[l,j]:11);
            end;writeln(list);
          end; writeln(list);
          writeln(list,'MINIMIZATION OVER ALL STAGES ');
          writeln(list);
          write(list,'N (stages)');
          for l:=1 to i do
            write(list,' GAIN:',i,' ');

          writeln(list);
          write(list,'----- ');
          for l:=1 to i do
            write(list,' -----');
          writeln(list);
          for j:= 1 to ti do
          begin
            write(list,' ',j:2,' ');
            for l:=1 to i do
            begin
              write(list,' ', graphArray1[j,l]:10);
              end;writeln(list);
            end;
          end;
        until ch in ['F','f','Q','q','P','p'];
        EnterGraphic;
        swapscreen;

```

```

close(list);
end;

begin
  clrscr; TextColor(lightblue);
  writeln('*** Design of Optimal Control
           procedure ***');
  TextColor(yellow);
  writeln('=====');
  gotoXY(1,25); InvVideo('Press <ESC> to change it!');
  Then Type your input with <ENTER> key');
  TextColor(green);
  Msg('Input number of time intervals for SUM
       procedure? ( MAX 99 ) ',1,4);
  input('N','50',65,4,6,true,F1,F10);
  val(answer,ti,code);
  Msg('What is your sample interval? ',1,6);
  input('N','0.1',35,6,6,true,F1,F10);
  val(answer,si,code);
  Msg('What is the value of scalar R ? ',1,8);
  Input('N','0.0',35,8,6,true,F1,F10);
  val(answer,r,code);
  Msg('For the following options Which cost
       function do you want ? ',1,10);
  TextColor(lightmagenta);
  Msg('COST=terminal+fuel+trajectory or
       COST=terminal+trajectory <0> ',1,11);
  Msg('COST=terminal+fuel
       COST=terminal <1> ',1,12);
  msg(' ',1,13); textcolor(yellow);
  msg('Where ',1,14); textcolor(lightmagenta);
  msg(' ',1,15);
  msg('Terminal = 1/2 XT(N) Q X(N) ',1,16);
  msg(' ',1,17);
  msg('Trajectory = 1/2  $\sum_{k=0}^{N-1} X^T(k) Q X(k)$  ',1,18);
  msg(' ',1,19);
  msg(' ',1,20);
  msg(' ',1,21);
  msg('Fuel = 1/2  $\sum_{k=0}^{N-1} U^T(k) R U(k)$  ',1,22);
  msg(' ',1,23);
  msg(' ',1,24);

  Input('N','0',65,10,2,true,F1,F10);
  val(answer,nterm,code); clrscr;
  writeln('enter the element of Q matrix: ');
  for i:=1 to size do
  begin
    for j:=1 to size do
    begin
      Q[i,j]:=0.0;
      write('Q( '); write(i); write(', '); write(j);
      write(')= '); readln(Q[i,j]);
    end; writeln;
  end;
  Repeat
    ClrScr; Writeln;
    Writeln('The Q Matrix is : '); writeln;
    for i:=1 to size do
    Begin
      for j:=1 to size do
      Begin
        Write(' ', Q[i,j]:11);
      End; Writeln;
    End; writeln;
    Write('Do you want to change any element of
           the Matrix ? ( Y / N ) ');
    Read(Kbd,Ans); writeln;
    (allows user to change entered data )
  
```

```

if ( Ans = 'Y') or ( Ans = 'y') then
Begin
  write('Input the row to change : '); readln(i);
  write('Input the column to change : ');
  readln(j); writeln;
  write('Q(,i,,j,')= '); readln(result);
  Q[i,j]:=result;
End;
Until Ans in ['N','n'];
textcolor(white);
Boxuser;
FI and GAMMA(SI,FI,GAMMA);
- ( call procedure to calculate fi and
                                     gamma matrices )
TextColor(yellow); clrscr;
writeln('The FI matrix is : ');
for i:=1 to size do
begin
  for j:=1 to size do
  begin
    write(' ',FI[i,j]:11);
  end;writeln;
end;writeln;writeln;
writeln('The gamma matrix is: ');
for i:=1 to size do
begin
  for j:=1 to ni do
  begin
    write(' ',GAMMA[i,j]:11);
  end;writeln;
end;writeln;writeln;
keyread(key,keyold,not_erased);
Boxuser;
for i:=1 to size do
begin
  D1[i]:=0.0;
  D2[i]:=0.0;
  for j:=1 to size do
  begin
    D3[i,j]:=0.0;
    D4[i,j]:=0.0;
    P1[i,j]:=Q[i,j];
  end;
end;
y:=0.0;ymax:=0.0;ymin:=0.0;
for kk:=1 to TI do
begin
  denominator:=0.0;
  for i:=1 to size do
  begin
    for j:=1 to size do
    begin
      D1[i]:=D1[i] + GAMMA[j,1] * P1[j,i];
    end;
  end;
  for i:=1 to size do
  begin
    for j:=1 to size do
    begin
      D2[i]:=D2[i] + D1[j] * FI[j,i];
    end;
    denominator:=denominator + D1[i] * GAMMA[i,1];
  end;
  denominator:=denominator+ R;
  negatif:=-1.0;
  for i:=1 to size do
  begin
    GT[i]:= negatif * D2[i] / denominator;

```

```

    GTN[I,KK]:=GT[I];
    D1[i]:=0.0;
    D2[i]:=0.0;
end;
for i:=1 to size do
begin
    for j:=1 to size do
    begin
        PSI[i,j]:=FI[i,j] + GAMMA[i,1] * GT[j];
    end;
end;
for i:=1 to size do
begin
    for j:=1 to size do
    begin
        for l:=1 to size do
        begin
            D3[i,j]:=D3[i,j] + PSI[l,i] * P1[l,j];
        end;
    end;
end;
for i:=1 to size do
begin
    for j:=1 to size do
    begin
        for l:=1 to size do
        begin
            D4[i,j]:=D4[i,j] + D3[l,1] * PSI[l,j];
        end;
        if nterm <= 0 then goto 2;
        P1[i,j]:=D4[i,j] + R * GT[i] * GT[j]; goto 3;
        2: P1[i,j]:=D4[i,j] + Q[i,j] + R * GT[i] * GT[j];
        3: D4[i,j]:=0.0;
    end;
end;
for i:=1 to size do
begin
    for j:=1 to size do
    begin
        D3[i,j]:=0.0;
    end;
end;
end;
for i:=1 to size do
begin
    for j:=1 to ti do
    begin
        GTNY[J]:=GTN[I,J];
        if abs(y) < 1.0e07 then y:=GTNY[j]
        else y:= 1.0e07 ;
        if y > Ymax then Ymax:=y;
        if Y < Ymin then Ymin:=Y;
        Grapharray[j,1] := j;
        grapharray[j,2] := y;
        grapharray1[j,1]:= y;
    end;
    Ymax:= 1.2 * Ymax;

    initgraphic;
    selectwindow(1);

    gotoxy(4,4);write('G');
    gotoxy(4,5);write('A');
    gotoxy(4,6);write('I');
    gotoxy(4,7);write('N');
    gotoxy(3,9);write(i:2);

```

```
drawtext(250,195,1,xarray);
Niceaxes(0,ti,ymin,ymax);
Selectworld(worldndxglb);
Selectwindow(windowndxglb);
Drawpolygon(grapharray,1,-(ti-1),0,0,0);

repeat until keypressed;
quit:=false;
repeat
  Graph_menu('OPTIMAL CONTROL GAIN
              PLOT',dumpgraph,quit);
  if dumpgraph then printgraphdata;
until quit;
leavegraphic;
end;

Assign(SysFile,'svs.COM');
Execute(SvsFile);
end.
```

```

program pole_placement(input,output);
{$I Typedef.sys}  {comman type & variable definitions}
{$I Ut-mod01.inc}
{$I Ut-mod02.inc}

label 100;label 110;label 120;label 130;label 140;
label 150;label 160;label 170;label 180;label 190;
label 200;label 210;

type
  ary6          = array [1..21] of integer;
  ary4          = array [1..11] of real;

var
  AA,PIN,P10          :ary1s;
  CCC                 :ary4;
  CC,H,DEN,E,HH       :ary4;
  realpart,imaginarypart,real_part,
  imaginary_part,realroot,imaginaryroot,
  realpart1,imaginarypart1 :ary3s;
  nn,umran,i,j,jj,k,l,n2,n1,m ,er
  vertpos,horizpos,poscounter,code :integer;
  gain1,test          :real;
  input2              :char;
  exponent,strg        :string[2];
  Change,factored      :boolean;
  specification        :string[5];
  inputfile            :file;
  reduceorder,msize,one :integer;
  list                 :text;

{$I Ut-mod03.inc}
{$I Pole.inc}
{$I Polynom.inc}
{$I Inverse.inc}
{$I Rootfind.inc}
{$I Boxuser.inc}

begin { open-loop calculations}
  nn:=size+1; Boxuser; luen:=false; one:=1;
  CHARACTERISTIC_EQUATION(A1A,SIZE,DEN);
  {call polynom.inc}
  clrscr;TextColor(yellow);
  writeln('Denominator of Y(s)/U(s) :');
  vertpos:=2;
  for i:=nn downto 1 do
    begin
      j:=nn-i;
      poscounter:=(j mod 4) +1;
      horizpos:=poscounter*14;
      if poscounter = 1 then vertpos:= vertpos+2;
      if i <> 1 then
        begin
          gotoXY(horizpos-10,vertpos);
          write(DEN[i]:7:4);
          msg('S +',horizpos,vertpos); str(i-1:2,exponent);
          msg(exponent,horizpos+1,vertpos-1);
        end
      else
        begin
          gotoXY(horizpos-10,vertpos);write(DEN[i]:7:4);
        end;
      end;writeln;writeln;writeln;writeln;
  root_finder(size,den,realpart1,imaginarypart1,one);

```

```

writeln('The poles of the Y(s)/U(s) are:');
write('REAL PART');
writeln('IMAGINARY PART ');
for i:=1 to size do
begin
  write(' ');
  write(realpart1[i]:7:4); write(' +j ');
  writeln(imaginarypart1[i]:7:4);
end;

keyread(key,keyold,not_erased);clrscr;

for i:= 1 to size do
  P10[i,size]:=B[i,1];

for jj:= 2 to size do
begin
  for i:=1 to size do
  begin
    j:=size-jj+1; k:=j+1;
    P10[i,j]:=DEN[k] * B[i,1];
    for l:=1 to size do
      P10[i,j]:= P10[i,j] + A1A[i,1] * P10[l,k];
    end;
  end;
  for i:=1 to size do
  begin
    CC[i]:=0.0;
    for l:=1 to size do
      CC[i]:= P10[j,i] * C[l,j] + CC[i];
    end;
  end;
  for i:=1 to size do
  begin
    m:=nn-i;
    if CC[m] <> 0.0 then goto 100;
  end;
  100:Textcolor(yellow);
  writeln('Numerator of Y(s)/U(s) :');
  vertpos:=2;
  for i:= m downto 1 do
  begin
    j:= m - i;
    poscounter:=(j mod 4) +1;horizpos:=poscounter*14;
    if poscounter = 1 then vertpos:= vertpos+2;
    if i <> 1 then
    begin
      gotoXY(horizpos-10,vertpos);write(CC[i]:7:4);
      msg('S +',horizpos,vertpos); str(i-1:2,exponent);
      msg(exponent,horizpos+1,vertpos-1);
    end
    else
    begin
      gotoXY(horizpos-10,vertpos);write(cc[i]:7:4);
    end;
  end;writeln;writeln;writeln;writeln;
  msize:=m-1;

  root_finder(msize,CC,realroot,imaginaryroot,one);

  writeln('The zeros of the Y(s)/U(s) are:');
  write('REAL PART');
  writeln('IMAGINARY PART ');
  for i:=1 to msize do
  begin
    write(' ');
    write(realroot[i]:7:4); write(' +j ');
    writeln(imaginaryroot[i]:7:4);
  end;

```

```

end;
keyread(key,keyold,not_erased);clrscr;
m:=m-1;
clrscr;textcolor(green);highvideo;
msg('Input the desired closed-loop characteristic
      equation ',1,6);
msg('Factored (F) form or Coefficient (C)
      form ?',1,7);

repeat read(kbd,input2)
until input2 in ['F','f','C','c'];
if (input2 = 'F') or (input2='f') then
begin
  if change then
    for j:=1 to size do
    begin
      str(real_part[j]:10:2,filvar[2*j+9]);
      str(imaginary_part[j]:10:2,filvar[2*j+10]);
    end;
    input_factored('POLES',size,real_part,
      imaginary_part);
    goto 120;
  end;
  if (input2='C') or (input2='c') then
  clrscr;m:=size+1;
  if change then
    for j:=m downto 1 do
    begin
      str(E[j]:10:2,Filvar[size+22-j]);
    end;umran:=1;
    input_coeff('POLES',size,e);

    root_finder(size,E,real_part,imaginary_part,one);
    clrscr;
    write('The roots of desired closed-loop
      characteristic');
    writeln(' polynomial are: ');
    write('          REAL PART');
    writeln('          IMAGINARY PART ');
    for i:=1 to size do
    begin
      write('
        write(real_part[i]:7:4); write('          +j          ');
        writeln(imaginary_part[i]:7:4);
      end;
      keyread(key,keyold,not_erased);clrscr;
      goto 180;

120:for i:=1 to size do
begin
  for j:=1 to size do
  begin
    AA[i,j]:=0.0;
  end;
end;
i:=0;
160:i:=i+1;
if imaginary_part[i] <> 0.0 then goto 150;
AA[i,i]:=real_part[i]; goto 170;
150:AA[i,i]:=real_part[i]; j:=i+1;
AA[i,j]:=-imaginary_part[i];
AA[j,i]:=-imaginary_part[i];
i:=j; AA[i,i]:=real_part[i];
170:if (i-size) < 0 then goto 160;

CHARACTERISTIC_EQUATION(AA,SIZE,E);

```

```

180:for i:=1 to size do
begin
  H[i]:= E[i] - DEN[i];
end;
for i:=1 to size do
begin
  CCC[i]:=C[1,i];
end;

INVERSE_FIND_MATRIX(P10,CCC,SIZE,PIN,CCC,ER);

for i:=1 to size do
begin
  HH[i]:=0.0;
  for j:=1 to size do
  begin
    HH[i]:= HH[i] + PIN[j,i] * H[j];
  end;
end;
for i:=1 to size do
begin
  for j:=1 to size do
  begin
    AA[i,j]:= A1A[i,j] - B[i,1] * HH[j];
  end;
end;

CHARACTERISTIC_EQUATION(AA,SIZE,E);

if CC[1]=0.0 then
begin
  gain1 :=1.0; goto 210;
end
else
gain1:= E[1] / CC[1] ;
210:if gain1 =0.0 then gain1 :=1.0;
textcolor(yellow);
for i:=1 to size do
begin
  HH[i]:= HH[i] /gain1;
end;
140:for i:=1 to size do
begin
  H[i]:=H[i]/gain1;
end;if H[size]=0.0 then reduceorder:=reduceorder-1;
clrscr;writeln('Numerator of the Heq(s) is :');
vertpos:=2;
for i:= size downto 1 do
begin
  j:= size -i;
  poscounter:=(j mod 4) +1;horizpos:=poscounter*14;
  if poscounter = 1 then vertpos:= vertpos+2;
  if i <> 1 then
  begin
    gotoXY(horizpos-10,vertpos);write(H[i]:7:4);
    msg('S +',horizpos,vertpos); str(i-1:2,exponent);
    msg(exponent,horizpos+1,vertpos-1);
  end
  else
  begin
    gotoXY(horizpos-10,vertpos);write(H[i]:7:4);
  end;
end;writeln;writeln;writeln;
reduceorder:=size-1;
root_finder(reduceorder,H,realpart,
             imaginarypart,one);
writeln('The roots of the Heq(s) are :');

```

```

write('
writeln('          IMAGINARY PART '); REAL PART');
for i:=1 to reduceorder do
begin
  write('
  write(realpart[i]:7:4); write('          ');
  writeln(imaginarypart[i]:7:4);
end;
keyread(key,keyold,not_erased); clrscr;
n2:=NN-1;
190:if H[n2] <> 0.0 then goto 200;
n2:=n2-1; goto 190;
200:n1:=n2-1;
TextColor(yellow); gotoXY(1,2);
writeln('The feedback coefficients [ k ] are : ');
writeln; for i:=1 to size do
begin
  write(' ');write(HH[i]:7:4);
end;writeln;writeln;
write('The gain [ K ] is :');
writeln(gain1:7:4);writeln;

keyread(key,keyold,not_erased);
Repeat
  box printer;textcolor(yellow);gotoxy(58,15);
  write('"POLE.RES" ');
  gotoxy(1,25);write(' ');
  gotoxy(49,17);
  Read(Kbd,ch);
  If (ch='P') or (ch='p') or (ch='F') or (ch='f')
  then Begin
    If (ch='F') or (ch='f') then
    Begin
      gotoxy(45,15);textcolor(red);
      write('PRINTING ');
      Assign(list,'pole.res');
      {print pole placement output }
      Rewrite(list);{to file on the current drive}
    End;
    If (ch='P') or (ch='p') then
    Begin
      gotoxy(45,13);textcolor(red);
      write('PRINTING ');
      Assign(list,'LST');
      {print pole placement result to the printer}

      Rewrite(list);
    End;
    Writeln(list,'          POLE PLACEMENT RESULT ');
    Writeln(list);Writeln(list);
    Writeln(list,'The Plant matrix A is : ');
    writeln(list);
    For i := 1 to size do
    Begin
      For j := 1 to size do
      Begin
        Write(List,' ',A1A[i,j]:11);
      End;Writeln(list);
    End;Writeln(list);Writeln(list);
    Writeln(list,'The Input matrix B is : ');
    writeln(list);
    For i := 1 to size do
    Begin
      For j := 1 to ni do
      Begin
        Write(List,' ',B[i,j]:11);
      End;Writeln(list);
    End;Writeln(list);Writeln(list);

```

```

Writeln(list,'The Output Matrix C is :');
writeln(list);
For i:= 1 to no do
begin
  for j:=1 to size do
  begin
    Write(list,' ',C[i,j]:11);
  end;writeln(list);
end;Writeln(List);Writeln(list);

write(list,'Denominator of Y(s)/U(s) -');
writeln(list,' Descending powers of S:');
writeln(list);
for i:=nn downto 1 do
begin
  write(list,' ',DEN[i]:7:4);
end;writeln(list);writeln(list);
writeln(list,'The poles of the Y(s)/U(s) are:');
write(list,' REAL PART');
writeln(list,' IMAGINARY PART ');
for i:=1 to size do
begin
  write(list,' ');
  write(list,realpart1[i]:7:4);
  write(list,' +j');
  writeln(list,imaginarypart1[i]:7:4);
end;writeln(list);writeln(list);

write(list,'Numerator of Y(s)/U(s) -');
writeln(list,' Descending powers of S:');
writeln(list);
for i:= m+1 downto 1 do
begin
  write(list,' ',CC[i]:7:4);
end;writeln(list);writeln(list);
writeln(list,'The zeros of the Y(s)/U(s) are:');
write(list,' REAL PART');
writeln(list,' IMAGINARY PART ');
for i:=1 to msize do
begin
  write(list,' ');
  write(list,realroot[i]:7:4);
  write(list,' +j');
  writeln(list,imaginaryroot[i]:7:4);
end;writeln(list);writeln(list);
write(list,'Desired closed-loop Characteristic
polynomial ');
writeln(list,'- Descending powers of S :');
writeln(list);
for i:= size+1 downto 1 do
begin
  write(list,' ',E[i]:7:4);
end;writeln(list);writeln(list);

write(list,'The roots of desired closed-loop
characteristic');
writeln(list,' polynomial are: ');
write(list,' REAL PART');
writeln(list,' IMAGINARY PART ');
for i:=1 to size do
begin
  write(list,' ');
  write(list,real_part[i]:7:4);
  write(list,' +j');
  writeln(list,imaginary_part[i]:7:4);
end;writeln(list);writeln(list);

```

```

Write(list,'Numerator of the Heq(s) is :');
writeln(list,'Descending powers of S :');
writeln(list);
for i:= size downto 1 do
begin
  write(list,'      ',H[i]:7:4);
end;writeln(list);writeln(list);
writeln(list,'The roots of the Heq(s) are :');
write(list,'      REAL PART');
writeln(list,'      IMAGINARY PART');
for i:=1 to reduceorder do
begin
  write(list,'      ');
  write(list,realpart[i]:7:4);
  write(list,'      ');
  writeln(list,imaginarypart[i]:7:4);
end;writeln(list);writeln(list);
writeln(list,'The feedback coefficients [ k ]
are :');
writeln(list); for i:=1 to size do
begin
  write(list,'      ');write(list,HH[i]:7:4);
end;writeln(list);writeln(list);
write(list,'The gain [ K ] is :');
writeln(list,gain1:7:4);
writeln(list);writeln(list);

Close(list);
End;
Until Ch in ['Q','q' ];
Assign(SysFile,'svs.COM');
      (re-execute SVS main program)
Execute(SvsFile);
End.

```

```

(*****
*          TURBO GRAPHIX version 1.00A          *
*          Type definition module                *
*          Copyright (C) 1985 by                 *
*          BORLAND International                  *
*****)

const MaxWorldsGlb=4;
      Maxorder=9;
      MaxWindowsGlb=16;
      MaxPiesGlb=10;
      MaxPlotGlb=200;
      MaxBlocks =10;
      StringSizeGlb=80;
      HeaderSizeGlb=10;
      RamScreenGlb:boolean=true;
      CharFile:string[StringSizeGlb]='4x6.fon';
      MaxProcsGlb=27;
      MaxErrsGlb=7;
      Extension:String[4]='.svs';

type wrkstring=string[StringSizeGlb];
      WorldType=record
          x1,y1,x2,y2:real;
      end;
      WindowType=record
          x1,y1,x2,y2:integer;
          header:wrkstring;
          drawn,top:boolean;
          size:integer;
      end;
      worlds=array [1..MaxWorldsGlb] of WorldType;
      windows=array [1..MaxWindowsGlb] of WindowType;
      PlotArray=array [1..MaxPlotGlb,1..2] of real;
      character=array [1..3] of byte;
      CharArray=array [32..126] of character;
      PieType=record
          area:real;
          text:wrkstring;
      end;
      PieArray=array [1..MaxPiesGlb] of PieType;
      BackgroundArray=array [0..7] of byte;
      LineStyleArray=array [0..7] of boolean;
      Ary1s=array [1..10,1..10] of real;
      Ary2s=array [1..10,1..10] of integer;
      Ary3s=array [1..10] of real;
      Str2=String[2];
      Str4=String[4];
      Str5=String[5];
      Str20=String[20];
      Str25=String[25];
      Str40=String[40];
      Str80=String[80];
      Str255=String[255];

var X1WldGlb,X2WldGlb,Y1WldGlb,Y2WldGlb,
    AxGlb,AyGlb,BxGlb,ByGlb:real;
    X1RefGlb,X2RefGlb,Y1RefGlb,Y2RefGlb:integer;
    LinestyleGlb,MaxWorldGlb,MaxWindowGlb,
    WindowNdxGlb:integer;
    X1Glb,X2Glb,Y1Glb,Y2Glb:integer;
    XTextGlb,YTextGlb,VStepGlb:integer;
    PieGlb,DirectModeGlb,ClippingGlb,
    AxisGlb,HatchGlb:boolean;
    MessageGlb,BrkGlb,HeaderGlb,TopGlb,
    GrafModeGlb:boolean;
    CntGlb,ColorGlb:byte;
    ErrCodeGlb:byte;

```

```
LineStyleArrayGlb:LineStyleArray;
ErrorProc:array [0..MaxProcsGlb] of ^WrkString;
ErrorCode:array [0..MaxErrsGlb] of ^WrkString;
PcGlb:string[40];
AspectGlb:real;
GrafBase:integer;
world:worlds;
window1:windows;
CharSet:CharArray;
D:Ary2s;
A1A,B,C:Ary1s;
Len,Space,Drive:Str2;
Size,ni,no,Block1,Key,Keyold:Integer;
Not_erased,Finished,Exit,Inserton,F1,F10:Boolean;

Ch:Char;
Template,Answer,Previous_Answer:Str80;
SvsFile:File;
P,Filvar:Array[1..35] of str40; {menu prompts}
worldndxglb :integer;
escape,retriev,luen : boolean;
```

```

procedure Bodeplot;
label 1; {label declaration for goto statement}
var
  Code,I,Count,NumberDecades,StartDecade,
  EndDecade : integer;
  Wf,Wo,Wi,DeltaW,Gain : Real;
  PlotArray1,PlotArray2,MagPhaseArray,
  FreqArray : PlotArray;
  ZMagn,ZPhase,PMagn,PPhase,Phase : real;
  TempX,TempY : real;
  temp : char;
  OpenLoop : boolean;
  j,jj,kk1,m,l,cnpoles,sizezeros,one : integer;
  denccoeff,numcoeff,cdenccoeff : ary4;
  realpartpole,imagpartpole,realpartzero,
  imagpartzero,crealpartpole,cimagpartpole: ary3s;
  PSI : ary1s;

function Log(X:real):real;
      {computes the base-10 logarithm of X}
Begin
  If X=0 then Log:=0 else
    Log := Ln(X)/Ln(10);
End;

function Expon(Y,X:real):real;
      {computes Y raised to X power}
Begin
  Expon := exp( X * (ln(Y)));
end;

begin
  ClrScr; boxuser; one:=1;
  Characteristic equation(A1A,size,Denccoeff);
  for i:=1 to size do
    begin
      PSI[i,size]:=B[i,1];
    end;
  for jj:=2 to size do
    begin
      for i:=1 to size do
        begin
          j:=size-jj+1;
          kk1:=j+1;
          PSI[i,j]:=Denccoeff[kk1] * B[i,1];
          for l:=1 to size do
            begin
              PSI[i,j]:=PSI[i,j] +A1A[i,l] * PSI[l,kk1];
            end;
          end;
        end;
      for i:=1 to size do
        begin
          Numcoeff[i]:=0.0;
          for j:=1 to size do
            begin
              Numcoeff[i]:=Numcoeff[i] + PSI[j,i] * C[1,j];
            end;
          end;
        end;
      for i:=1 to size do
        begin
          m:=size+1-i;
          if numcoeff[m] <> 0.0 then goto 1;
        end;
      1:sizezeros:=m-1;
      ClrScr;TextColor(lightblue);
      writeln(' *** Bode Plotting Parameters ***');
      TextColor(yellow);

```

```

writeln('=====');
Msg('Open (O) or Closed (C) Loop Plot?',5,5);
repeat
  input('A','',45,5,2,true,F1,F10);
  {sets flag OpenLoop if}
  temp := copy(answer,1,1);
  {user selects the open}
  if not(temp in ['O','C']) then beep(350,150);
until temp in ['O','C'];
if (temp = 'O') then OpenLoop := true
else OpenLoop := false;
Msg('What is the first frequency to be
  plotted?',5,7);
Msg('example: .01, 1, 100, etc.','','10,8);
Input('N','',50,7,8,true,F1,F10);
Val(answer,Wo,code); {Wo is the first plotted freq}

Msg('Input number of decades do you want
  plotted?',5,10);
Input('N','',51,10,2,true,F1,F10);
Val(answer,NumberDecades,code);

root_finder(sizezeros,Numcoeff,realpartzero,
  imagpartzero,one);
root_finder(size,Dencoeff,realpartpole,
  imagpartpole,one);

gain:= Numcoeff[sizezeros+1];
for i:=1 to sizezeros+1 do
begin
  Numcoeff[i]:= Numcoeff[i]/gain;
end;
boxuser;

for i:=1 to maxorder do CDenCoeff[i] := 0.0;
for i:=1 to SizeZeros + 1 do
  CDenCoeff[i] := Numcoeff[i] * gain;

for i:=1 to Size + 1 do
  CDenCoeff[i] := CDenCoeff[i] + Dencoeff[i];

if Size > SizeZeros then CNPOLES:=Size
  {NPoles should always be}
else CNPOLES:=SizeZeros;
{greater,but to be safecompute new denominator roots}

root_finder(Cnpoles,CDenCoeff,CRealPartPole,
  CImagPartPole,one);

StartDecade := trunc(Log(Wo));
EndDecade := StartDecade + NumberDecades;
{compute linear scale to plot }
{log numbers. Also figure step}
Wf := Wo * Expon(10.0,NumberDecades);
DeltaW := Expon((Wf/Wo),0.0125);
Wi := Wo;

for Count := 1 to 81 do{do 81 iterations...arbitrary}
Begin
  if OpenLoop then{compute bode numbers if openloop}
  {and later if closed loop}
  begin

```

```

ZMagn:=1.0; ZPhase:=0.0; PMagn:=1.0; PPhase:=0.0;
for i := 1 to SizeZeros do
  {compute magn and phase of zeros for freq step}
  begin
    ZMagn:=ZMagn * Sqrt(Sqr(RealPartZero[I])+
      Sqr(Wi-ImagPartZero[I]));
    if RealPartZero[I] = 0.0 then
      ZPhase:=ZPhase+pi/2.0 else;
    begin
      if realpartzero[i] > 0.0 then
        ZPhase:=ZPhase - pi +arctan((wi-
          imagpartzero[i])/(-realpartzero[i]))
      else
        ZPhase:=ZPhase + arctan((wi-
          imagpartzero[i])/(-realpartzero[i]));
    end;
  end;

  for i := 1 to Size do
    {compute magn and phase of poles for freq step}
    begin
      PMagn:=PMagn * Sqrt(Sqr(RealPartPole[I]) +
        Sqr(Wi-ImagPartPole[I]));
      if RealPartPole[I] = 0.0 then
        PPhase:= PPhase+pi/2.0 else
        begin
          if realpartpole[i] > 0.0 then
            PPhase:=PPhase - pi +arctan((wi-
              imagpartpole[i])/(-realpartpole[i]))
          else
            PPhase:=PPhase + arctan((wi-
              imagpartpole[i])/(-realpartpole[i]));
        end;
    end;

    PlotArray1[Count,1] := Log(Wi);
    {fill plotting matrix with magnitude values}
    PlotArray1[Count,2] := 20*Log(Gain*
      (ZMagn/PMagn));

    PlotArray2[Count,1]:=Log(Wi);{fill phase matrix}
    PlotArray2[Count,2] := (180/pi)*(ZPhase-PPhase);

    {next stmt covers freq wrap-around}
    if PlotArray2[Count,2] > 0 then
      PlotArray2[count,2]:=PlotArray2[count,2]-360;

    Wi := Wi * DeltaW;          {increment freq step}
  end

  else
    {perform same steps as above if closed loop requested}
    begin
      ZMagn:=1.0; ZPhase:=0.0; PMagn:=1.0; PPhase:=0.0;
      for i := 1 to SizeZeros do
        begin
          ZMagn:=ZMagn * Sqrt(Sqr(RealPartZero[I])+
            Sqr(Wi-ImagPartZero[I]));
          if RealPartZero[I] = 0.0 then
            ZPhase:=ZPhase+pi/2.0 else
            begin
              if Realpartzero[i] > 0.0 then

```

```

        ZPhase:=ZPhase - pi + arctan((wi-
            imagpartzero[i])/(-realpartzero[i]))
    else
        ZPhase:=ZPhase + arctan((wi-
            imagpartzero[i])/(-realpartzero[i]));
    end;
end;
for i := 1 to CNpoles do
begin
    PMagn:=PMagn * Sqrt(Sqr(CRealPartPole[I])+
        Sqr(Wi-CImagPartPole[I]));
    if CRealPartPole[I] = 0.0 then
        PPhase:=PPhase+pi/2.0 else
    begin
        if crealpartpole[i] > 0.0 then
            PPhase:=PPhase - pi + arctan((wi-
                cimagpartpole[i])/(-crealpartpole[i]))
        else
            PPhase:=PPhase + arctan((wi-
                cimagpartpole[i])/(-crealpartpole[i]));
        end;
    end;
    PlotArray1[Count,1] := Log(Wi);
    {fill plotting matrix with magnitude values}
    PlotArray1[Count,2] := 20*Log(Gain *
        (ZMagn/PMagn));
    PlotArray2[Count,1]:=Log(Wi);{fill phase matrix}
    PlotArray2[Count,2] := (180/pi)*(ZPhase-PPhase);
    {next stmt covers freq wrap-around}
    if PlotArray2[Count,2] > 0 then
        PlotArray2[count,2]:=PlotArray2[count,2]-360;
    Wi := Wi * DeltaW;
end;
end;
PlotBode(StartDecade,EndDecade,NumberDecades,
    PlotArray1,PlotArray2,OpenLoop);
end; {bodeplot}

```

132

```

    assign(list,'Bode.RES');
    rewrite(list);
end
else
begin
    gotoxy(24,13);textcolor(red);
    write('PRINTING.....' );

    assign(list,'LST:');
    rewrite(list);
end;

Title1:=( '          w (rad)          Gain (db)
          Phase (deg)');
Title2:=( '-----' );

writeln(list,'          BODE PLOT RESULT ');

writeln(list);writeln(list);
writeln(list,Title1);
writeln(list,Title2);
writeln(list);writeln(list);
for i:= 1 to 81 do
begin
    w:= expon(10,0,PlotArray1[i,1]);
    writeln(list,w:11:3,PlotArray1[i,2]:8:3,PlotArray2[i,2]:7:3);
    if i= 47 then
    begin
        write(list,chr(12));
        writeln(list);
        writeln(list,Title1);
        writeln(list,Title2);
        writeln(list);writeln(list);
    end;
end;
end;
until ch in ['Q','q'];
EnterGraphic;
{when finished printing, go back to graphics mode
and display graph}
swapscreen;
close(list);
end;

Begin
initgraphic; {set-up windows for display}
DefineWindow(1,0,0,XMaxGlb,YMaxGlb);
DefineWindow(2,5,15,XMaxGlb-5,YMaxGlb-15);
DefineWindow(3,5,15,XmaxGlb-5,YMaxGlb-15);
DefineWorld(1,0,0,100,100);
DefineWorld(2,StartDecade,60,EndDecade,-60);
DefineWorld(3,StartDecade,0,EndDecade,-360);
SelectWorld(1);
SelectWindow(1);
SetBackground(0);
SelectWorld(2);
SelectWindow(2);
DrawBorder;

SetLineStyle(1);
For I:=1 to 5 do {draw horizontal graph lines}
    DrawLine(StartDecade,-60+(20*I),
              EndDecade,-60+(20*I));
For J:=0 to NumberDecades-1 do
    {draw vertical logarithmic graph lines}
Begin

```

```

For I:= 1 to 10 do
  Begin
    Delta:=StartDecade + (Log(I) + J);
    Drawline(Delta,-60,Delta,60);
  end;
end;
SelectWindow(1);           {y-axis titles}
For I:= 1 to 12 do
  Begin
    DrawText(5,55+6*I,1,MagnArray[I]);
    DrawText(630,60+6*I,1,PhasArray[I]);
  end;
DrawText(250,195,1,FreqArray);{x-axis title}
For I:= 0 to 6 do          {y-axis scale label}
  Begin
    Str(60-20*I:3,MagLabel);
    DrawText(12,13+28*I,1, MagLabel);
    Str(0-60*I:4,PhsLabel);
    DrawText(600,13+28*I,1,PhsLabel);
  end;
For I:= 0 to NumberDecades do
  {label the logarithmic scale}
  Begin
    Str(Trunc(StartDecade)+I:3,DecLabel);
    DrawText(36+(570 div NumberDecades)*I,
              186,1,DecLabel);
    DrawText(30+(570 div NumberDecades)*I,
              190,1,'10');
  end;
SetLineStyle(0);
SelectWindow(2); SelectWorld(2);
DrawPolygon(PlotArray1,1,-81,0,1,0);
              {plot the magnitude}
SelectWorld(3);
SelectWindow(3);
SetLineStyle(3);
DrawPolygon(PlotArray2,1,-81,0,1,0);
              {plot the phase}
copyscreen;           {save screen to memory}
repeat until keypressed;
quit := false;
repeat {call for graph options menu}
  if OpenLoop then
    Graph_Menu('Open Loop Bode Plot',DumpGraph,quit)
  else
    Graph_Menu('Closed Loop Bode',DumpGraph,quit);
  If DumpGraph then PrintGraphData;
                    {dump numbers if desired}
until quit;
LeaveGraphic;         {leave graphics mode}
end;

```

```
(* This program draw line for menu box. *)
procedure box;
var i :integer;
begin
  Highvideo;TextColor(yellow);gotoxy(1,2);
  write(chr(218)); {draw upper left corner}
  for i := 1 to 77 do {draw upper horizontal line}

  begin
    write(chr(196));
  end; write(chr(191));
  gotoxy(1,5); write(chr(192)) ;
  for i:= 2 to 78 do
    write(chr(196));
  write(chr(217));
  for i:= 3 to 4 do
  begin
    gotoxy(1,i);write(chr(179)); {draw vertical lines}

    gotoxy(79,i);write(chr(179));
  end;
  gotoxy(1,23);
  for i:= 1 to 78 do
    write(chr(196));
  writeln(',');
  TextColor(white);gotoXY(20,22);
  writeln('Make your selection');
  gotoxy(40,22);Lowvideo; {make input prompt}
end;
```

```

Procedure Fi_and_gamma(var T:real;
                        var fi:ary1s;
                        var gamma:ary1s );
label 1;
var
  A2,A3,A4,A5                :ary1s;
  i,j,k                      :integer;
  test_value,step            :real;
begin
  test_value:=1.0e-08;
  step:=1.0;
  for i:=1 to size do
  begin
    for j:=1 to size do
    begin
      FI[i,j]:=0.0;
      FI[i,i]:=1.0;
      A4[i,j]:=A1A[i,j];
      A2[i,j]:=(T/2.0)*FI[i,j];
      A5[i,j]:=T*FI[i,j];
    end;
  end;
  1:for i:=1 to size do
  begin
    for j:=1 to size do
    begin
      A3[i,j]:=(T/step)*A4[i,j];
      FI[i,j]:=FI[i,j]+A3[i,j];
      A2[i,j]:=A2[i,j]+(T/((step+1.0)*(step+2.0)))*A3[i,j];
      A5[i,j]:=A5[i,j]+(T/(step+1.0))*A3[i,j];
    end;
  end;
  for i:=1 to size do
  begin
    for j:=1 to size do
    begin
      A4[i,j]:=0.0;
      for k:=1 to size do
      A4[i,j]:=A4[i,j]+A1A[i,k]*A3[k,j];
      end;
    end;
    step:=step+1.0;
    for i:=1 to size do
    for j:=1 to size do
      if abs(A3[i,j])>(test_value*abs(FI[i,j]))
      then goto 1;

    for i:=1 to size do      {initialize}
    for j:=1 to ni do
      GAMMA[i,j]:=0.0;

    for i:=1 to size do      {calculate GAMMA matrix }
    for j:=1 to ni do
    for k:=1 to size do
      GAMMA[i,j]:=GAMMA[i,j]+A5[i,k]*B[k,j];
    end;
  end;

```

```

Procedure Matrix_reduction(Var dummy_square:Ary1;
                           Var n,m,rank:Integer);

```

```

Label 142;Label 145;Label 235;Label 170;Label 180;
Label 162;Label 198;Label 200;Label 224;Label 226;
Label 228;Label 234;

```

```

VAR

```

```

    reduction_matrix           : Ary1;
    Jjc,Nmin,i,j,jj,kj,k,j1,start,      : Integer;
    l,l1,imin,l1,l11,jk,l1            : Real;
    compare_value,temp_value,sum

```

```

Begin

```

```

    For i := 1 to n do
        For j:=1 to m do
            reduction_matrix[i,j] := dummy_square[i,j];

```

```

    compare_value := 1.00E-8;

```

```

    Jjc :=1;

```

```

    Nmin := n - 1;

```

```

    If nmin = 0 then goto 235;

```

```

    For i := 1 to nmin do

```

```

        Begin

```

```

            For j := jjc to m do

```

```

                Begin

```

```

                    Jj :=j;

```

```

                    For k := i to n do

```

```

                        Begin

```

```

                            Kj :=k;

```

```

                            If abs( reduction_matrix[k,j]) >=compare_value
                                then GoTo 142;-

```

```

                        End;

```

```

                    End;

```

```

                    GoTo 235;

```

```

                    142:If kj = i then GoTo 145;

```

```

                    For j1 := jj to m do

```

```

                        Begin

```

```

                            Temp_value := reduction_matrix[i,j1];

```

```

                            reduction_matrix[i,j1]:=reduction_matrix[kj,j1];

```

```

                            reduction_matrix[kj,j1] :=temp_value;

```

```

                        End;

```

```

                    145:temp_value := reduction_matrix[i,jj];

```

```

                    For jk := jj to m do

```

```

                        reduction_matrix[i,jk] := reduction_matrix[i,jk]
                                                /temp_value;

```

```

                    If i <>1 then GoTo 180;

```

```

                    start :=2;

```

```

                    162:for l := start to n do

```

```

                        Begin

```

```

                            Temp_value := reduction_matrix[l,jj];

```

```

                            If abs( reduction_matrix[l,jj]) <= compare_value
                                then GoTo 170;-

```

```

                            For l1 :=jj to m do

```

```

                                reduction_matrix[l,l1]:=reduction_matrix[l,l1]-
                                    temp_value * reduction_matrix[l,l1];

```

```

                    170:End;

```

```

                    GoTo 200;

```

```

                    180:imin :=i-1;

```

```

                    For l1 :=1 to imin do

```

```

                        Begin

```

```

                            Temp_value := reduction_matrix[l1,jj];

```

```

                            If abs( reduction_matrix[l1,jj]) <=compare_value
                                then GoTo 198;-

```

```

    For lli := jj to M do
      reduction_matrix[li,lli] :=
        reduction_matrix[li,lli] - reduction_matrix[i,lli]
        * temp_value;

198 :End; start := i + 1;
GoTo 162;
200: If jj = m then goto 235;
    Jjc := jj + 1;
End;
For jk := jj to m do
  Begin
    Temp_value := reduction_matrix[n,jk];
    Jjc := jk;
    If abs(reduction_matrix[n,jk]) <= compare_value
      then GoTo 226;
    For j1 := jk to m do
      reduction_matrix[n,j1] := reduction_matrix[n,j1] /
        temp_value;

    GoTo 228;
226: End;
GoTo 235;
228: For lf := 1 to nmin do
  Begin
    Temp_value := reduction_matrix[lf,jjc];
    If abs(reduction_matrix[lf,jjc]) <= compare_value
      then GoTo 234;
    For j1 := jjc to m do
      reduction_matrix[lf,j1] := reduction_matrix[lf,j1]
        - temp_value *
        reduction_matrix[n,jjc];

234: End;
235: Rank := 0;
    For i := 1 to n do
      Begin
        Sum := 0.0;
        For j := 1 to m do
          Begin
            If abs(reduction_matrix[i,j]) <= compare_value
              then reduction_matrix[i,j] := 0.0;
            dummy_square[i,j] := reduction_matrix[i,j];
            Sum := sum + abs(reduction_matrix[i,j]);
          End;
        If sum >= compare_value then rank := rank + 1;
      End;
    End;

  Procedure matrix_multiplication(VAR matrix, matrix1,
                                   matrix2: Ary1s;
                                   VAR L,M,N : Integer);

  var mult_matrix1, result_matrix, mult_matrix2 : Ary1s;
      k,j,i : Integer;

  Begin
    For j:=1 to m do
      begin
        For i :=1 to l do
          begin
            mult_matrix1[i,j] := matrix[i,j];
          end;
          For k:=1 to n do
            mult_matrix2[j,k] := matrix1[j,k];
          End;
          For i :=1 to l do

```

```
Begin
  For j:=1 to n do
    Begin
      result_matrix[i,j] := 0.0;
      For k:=1 to m do
        result_matrix[i,j] := result_matrix[i,j] +
          mult_matrix1[i,k] * mult_matrix2[k,j];
      End;
    End;
  For i:= 1 to l do
    Begin
      For j:=1 to n do
        matrix2[i,j] := result_matrix[i,j];
      End;
    End;
  End;
```

```

procedure Inverse_find_matrix(VAR A:Ary1s;
                               var XDOT:ary4;
                               var ORDER:integer;
                               var AINV:ary1s;
                               var X:ary4;
                               var singular:integer);

label 5;label 6;label 12;label 16;label 20;label 51;

var
  B2                                :ary1s;
  comp,temporary_value              :real;
  n,i,j,k,m                        :integer;

begin
  n:=1;
  for i:=1 to order do
    begin
      for j:=1 to order do
        begin
          AINV[i,j]:=0.0;
          B2[i,j]:=A[i,j];
        end;
      end;
      for i:= 1 to order do
        begin
          AINV[i,i]:=1;
          X[i]:=XDOT[i];
        end;
        for i:=1 to order do
          begin
            comp:=0.0;
            k:=1;
            6:if (abs (B2[k,i]) -abs(comp)) <= 0.0 then goto 5;

            comp:=B2[k,i];
            n:=k;
            5:k:=k+1;
            if (k-order) <= 0 then goto 6;
            if B2[n,i] = 0.0 then goto 51;
            if (n-i) < 0 then goto 51;
            if (n-i) = 0 then goto 12;
            for m:=1 to order do
              begin
                temporary_value:= B2[i,m];
                B2[i,m] :=B2[n,m];
                B2[n,m] :=temporary_value;
                temporary_value:= AINV[i,m];
                AINV[i,m]:= AINV[n,m];
                AINV[n,m]:= temporary_value;
              end;
              temporary_value:=X[i];
              X[i]:=X[n];
              X[n]:=temporary_value;
              12:X[i]:=X[i] /-B2[i,i];
              temporary_value:=B2[i,i];
              for m:=1 to order do
                begin
                  AINV[i,m]:= AINV[i,m] / temporary_value;
                  B2[i,m]:=B2[i,m] / temporary_value;
                end;
                for j:=1 to order do
                  begin
                    if (j-i) = 0 then goto 16;
                    if B2[j,i] = 0.0 then goto 16;
                    X[j]:=X[j] - B2[j,i] * X[i];
                    temporary_value:=B2[j,i];
                    for n:=1 to order do

```

```
begin
  AINV[j,n]:= AINV[j,n] - temporary_value *
    AINV[i,n];
  B2[j,n]:=B2[j,n] - temporary_value * B2[i,n];
end;
16:end;
end;
goto 20;
51:singular:=1;
20:end;
```

```

Procedure linear_equation(var AA:ary2;
                        var nw,m:integer;
                        var BB,x:ary3;
                        var K:integer;
                        var U:ary2);

label 1;label 2;label 3;label 4;label 5;label 71;
label 81;label 120;label 8;label 6 ;label 10;label 130;
label 7;

Var
  is,it,kk,n,k1,ii,mm,i,j           :integer;
  a1                                 :ary2;
  id                                 :ary3;
  b1,c1,ic,w                        :real;

Begin {linear equation}
  n:=nw;
  mm:=m+1;
  for i:=1 to n do
    Begin
      A1[i,mm] := BB[i];
      for j:=1 to m do
        A1[i,j] := AA[i,j];

      End;
      k:=1;
      if (n-m) >= 0 then goto 1;
      it:= n+1;
      n:=m;
      for i:= it to m do
        Begin
          for j:=1 to mm do
            A1[i,j] := 0.0;

          End;
          1:for i:=1 to m do
            ID[i]:= i;

          2:kk:=k+1;
          is:=k;
          it:=k;
          B1:=abs(A1[k,k]);
          for i:=k to n do
            Begin
              for j:=k to m do
                Begin
                  if( abs(A1[i,j])-B1) <= 0 then goto 3;
                  is:=i;
                  it:= j;
                  B1:=abs(A1[i,j]);
                End;
              3:End;
            End;
            if(is-k) <= 0 then goto 4;
            for j:= k to mm do
              Begin
                c1:= A1[is,j];
                A1[is,j] := A1[k,j];
                A1[k,j] := c1;
              End;
            4:if(it-k) <=0 then goto 5;
            ic:=ID[k];
            ID[k] :=ID[it];
            ID[it] :=ic;
            for i:= 1 to n do
              Begin
                C1:=A1[i,it];
                A1[i,it] := A1[i,k];

```

```

    A1[i,k] := c1;
End;
5: if A1[k,k] <> 0 then goto 71;
kk:=k;
k:=k-1;
for i:=kk to m do
    A1[j,j] := -1.0;

goto 6;
71: if (k-n) > 0 then goto 120;
if (k-n) < 0 then goto 81;
A1[n,mm] := A1[n,mm] / A1[n,n];
goto 7;
81: for j:=kk to mm do
Begin
    A1[k,j] := A1[k,j] / A1[k,k];
    for i:=kk to n do
    Begin
        w:=A1[i,k] * A1[k,j];
        A1[i,j] := A1[i,j] - w;
        if (abs(A1[i,j]) - 0.0001*abs(w)) >= 0
        then goto 8;
        A1[i,j] := 0.0;
    End;
8: End;
End;
if (k-m) > 0 then goto 120;
if (k-m) = 0 then goto 6;
k:=kk;
goto 2;
6: for i:=kk to n do
    if A1[i,mm] <> 0 then goto 120;

7: k1:=k-1;
for is:=1 to k1 do
Begin
    i:=k-is;
    ii:=i+1;
    for it:=ii to k do
    Begin
        for j:=kk to mm do
            A1[i,j] := A1[i,j] - A1[i,it] * A1[it,j];
    End;
End;
for i:=1 to m do
Begin
    for j:=1 to m do
    Begin
        if (ID[j]-i) <> 0 then goto 10;
        X[i] := A1[j,mm];
        if (k-m) = 0 then goto 10;
        for is:=kk to m do
            U[i,is-k] := A1[j,is];
    End;
10: End;
End;
k:=m-k;
goto 130;
120: Writeln;
Writeln('There are no equations'); Delay(2000);
130: End; {linear equation}

```

```

Procedure Polynomial_of_roots(var n:integer;
                             var rr,ri:ary3s;
                             var cf:ary4);
(This program calculates the coefficients of a
 polynomial from its roots)
label 1;label 3;label 111;label 222;label 2;
label 5;label 21;label 50;

var
  ick,nn,mp,i,ii,l,m,mm,mmp      :integer;
  sumreal,prreal1,sumimag,prreal,primag :real;
  j                                :ary6;

Begin
  nn:=n+1;
  CF[nn]:=1.0;
  for m:=1 to n do
    Begin
      sumreal:=0.0;
      sumimag:=0.0;
      l:=1;
      J[l]:=1;
      goto 2;
      1:J[l]:=J[l]+1;
      2:if (l-m) > 0 then goto 50;
      if (l-m) = 0 then goto 5;
      mm:=m-1;
      for i:=1 to mm do
        Begin
          ii:=i+1;
          J[ii]:=J[i]+1;
        End;
      5:prreal:=1.0;
      prreal1:=1.0;
      primag:=0.0;
      for i:=1 to m do
        Begin
          prreal:=prreal1;
          ick:=J[i];
          prreal1:= prreal*RR[ick] - primag*RI[ick];
          primag:= prreal*RI[ick]+primag*RR[ick];
        End;
      sumreal := sumreal+prreal1;
      sumimag:= sumimag+primag;
      for i:=1 to m do
        Begin
          l:=m-i+1;
          if (J[l]-n+m-1) > 0 then goto 50;
          if (J[l]-n+m-1) < 0 then goto 1;
        End;
      mp:=n-m+1;
      mmp:=mp mod 2;
      if mmp <> 0 then goto 111;
      CF[mp]:= sumreal;goto 222;
      111: CF[mp]:=-sumreal;
      222:End;
      goto 21;
      50:Writeln;
      Writeln('There is an error in polynomial
               calculation from roots procedure');
      Delay(2000);
      21:mmp:= n mod 2;
      if mmp <> 0 then goto 3;
      for i:= n downto 1 do
        CF[i]:=-CF[i];
      3:End;

```

{Plot Nyquist is a routine to draw the Nyquist plot
from the data generated in the Nyquist procedure. }

```
Procedure Plot_Nyquist(StartDecade, EndDecade,
                      NumberDecades: Integer;
                      FreqArray, PlotArray1,
                      Plotarray2,
                      MagPhaseArray: PlotArray;
                      BigPic, OpenLoop: Boolean);
```

```
var
  Xint, Yint, i, j, n, code : integer;
  Ymin, Ymax, Xmin, Xmax   : real;
  XLabel, YLabel          : String[3];
  Title1, Title2          : String[80];
  DumpGraph, quit        : Boolean;
  w, XminL, YminL         : Real;
  GraphWidthX, GraphWidthY: Real;
  Xexponent, Yexponent    : integer;
  XexpLabel, YexpLabel    : string[3];
  GraphArray              : PlotArray;
  List                    : text;
```

```
function Expon(Y, X: real): real;
      {computes Y raised to X power}
Begin
  Expon := exp( X * (ln(Y)));
end;
```

```
Procedure PrintGraphData;
      {dump data used to make graph to printer}
Begin
  LeaveGraphic; Clrscr;
  repeat
    Textcolor(white); gotoxy(20, 10);
    writeln('*** PROGRAM OUTPUT OPTIONS *** ');
    gotoxy(20, 13); writeln('<P> Printer output ');
    Textcolor(yellow); gotoxy(20, 14);
    writeln('Check Your Printer! ');
    Textcolor(white); gotoxy(20, 15);
    writeln('<F> List to File name ');
    gotoxy(20, 16);
    writeln('on the current drive ');
    gotoxy(20, 17);
    writeln('<Q> Quit ');
    gotoxy(42, 15);
    textcolor(yellow); write('"NYQUIST.RES"');
    gotoxy(28, 17);
    read(kbd, ch);
    If (ch = 'F') or (ch = 'f') or (ch = 'P') or
       (ch = 'p') then
      begin
        if (ch = 'F') or (ch = 'f') then
          begin
            gotoxy(24, 15); textcolor(red);
            write('PRINTING..... ');
            assign(list, 'Nyquist.RES');
            rewrite(list);
          end
        else
          begin
            gotoxy(24, 13); textcolor(red);
            write('PRINTING..... ');
            assign(list, 'LST:');
            rewrite(list);
          end
        end
      end
    end
  repeat
```

```

end;
Title1:=( '      w (rad)      Magnitude
          Phase (rad)      Xplot      YPlot');
Title2:=( '-----');
writeln(list,'      NYQUIST PLOT RESULT');
writeln(list);writeln(list);
writeln(list,Title1); writeln(list,Title2);
writeln(list); writeln(list);
for i:= 1 to 81 do
begin
  w:= FreqArray[i,1];
  writeln(list,w:9:3,
           MagPhaseArray[i,1]:11:3,'
           MagPhaseArray[i,2]:11:3,'
           PlotArray1[i,1]:10:3,'
           PlotArray1[i,2]:10:3);
  if i= 47 then
  begin
    writeln(list);
    write(list,chr(12));
    writeln(list,Title1);
    writeln(list,Title2);
    writeln(list); writeln(list);
  end;
end;
end;
until ch in ['Q','q'];
EnterGraphic;
swapscreen;
close(list);
end;

```

```

Begin (Plot_Nyquist) (prompt for window parameters)
if not(BigPic) then
begin
  P[3]:= '1511N00503-010101';
  P[4]:= '1512N00504-010101';
  P[5]:= '1513N00505-010101';
  P[6]:= '1514N00506-010101';

  Clrscr; TextColor(LightBlue);
  writeln(' *** NYQUIST PLOTTING PARAMETERS ***');
  textcolor(yellow);
  writeln('=====');

  HighVideo;
  writeln;writeln;Textcolor(white);
  msg('Input Plotting Coordinates for the
      Nyquist Plot',1,6);
  TextColor(yellow);
  writeln;writeln;writeln;writeln;
  writeln('X-Minimum: ');
  writeln('X-Maximum: ');
  writeln('Y-Minimum: ');
  writeln('Y-Maximum: ');

  Input_Handler('N0306',Escape);
  writeln;writeln;
  writeln('Any changes to these parameters?
      (Y / N):');
  Input('A','',45,16,2,true,F1,F10);
  If (answer='Y') or (F1) then
    Input_Handler('C0306',Escape);
  Val(filvar[3],Xmin,code);
  Val(filvar[4],Xmax,code);

```

```

    Val(filvar[5],Ymin,code);
    Val(filvar[6],Ymax,code);
end
else
begin
    Xmax:= 100;
        {set default values for "big picture" plot}
    Xmin:= -100;
    Ymax:= 100;
    Ymin:= -100;
end;

INITGRAPHIC;
niceaxes(xmin,xmax,ymin,ymax, '');
n:=1;
for i:=1 to 80 do
if (abs(plotarray1[i,1]) > Xmax) or
   (abs(plotarray1[i,2]) > Ymax) then    n:= n+1;
if n<>1 then n:= n-1;
        {use i extra point beyond graph border}

DrawPolygon(PlotArray1,n,-80,0,1,0);
        {draw graph on screen}
n:=1;
for i:=1 to 80 do
if (abs(plotarray2[i,1]) > Xmax) or
   (abs(plotarray1[i,2]) > Ymax) then

n:= n+1;
if n<>1 then n:= n-1;
Setlinestyle (1);
DrawPolygon(Plotarray2,n,-80,0,1,9);

Repeat until Keypressed;
        {Put option menu on screen}
quit:= false;
repeat
if OpenLoop then
    Graph_Menu('Open Loop NyquistPlot',
        DumpGraph,quit)
else
    Graph_Menu('Closed Loop Nyquist Plot',
        DumpGraph,quit);
if DumpGraph then PrintGraphData;
until quit;
LeaveGraphic;
end;

```

{Graph_Menu provides a window on screen and offers the user options to make a title, print the graph, print the numbers from the graph, or quit and return to the menu. }

```
Procedure Graph_Menu(TitleWindowName:STR25;
                     var DumpGraphData,quit : boolean);
```

```
var
  Line1, Line2, Line3 : string[40];
```

```
Procedure TitlePrompt;
```

```
begin
  TextColor(White);
  Center('*** Graph Title ***',1,2,80);

  P[1] := '1010A04001-010100';
  P[2] := '1012A04002-010100';
  P[3] := '1014A04003-010100';

  textcolor(yellow);
  msg('Line 1:',1,10);
  msg('Line 2:',1,12);
  msg('Line 3:',1,14);

  textcolor(green);
  msg('Type your title for your graph.',6,20);

  Input_handler('N0103',escape);

  Line1:= copy(filvar[1],1,40);
  Line2:= copy(filvar[2],1,40);
  Line3:= copy(filvar[3],1,40);
```

```
end;
```

```
Procedure ShowTitle;
  {makes title block and writes title to block}
```

```
begin
  copyscreen;
  SetLineStyle(0);
  DefineWindow(3,11,20,40,60);
  DefineWorld(3,0,0,40,16);
  SelectWorld(3); SelectWindow(3);
  DefineHeader(3,TitleWindowName); {puts header on box}

  SetBackground(0);
  SetHeaderOn; DrawBorder;
  DrawTextW(1,4,1,Line1);
  DrawTextW(1,8,1,Line2);
  DrawTextW(1,12,1,Line3);
  SetBreakOff; SetMessageOff;

  repeat
    read(kbd,ch);
    case ord(ch) of
      {allow user to move title box anywhere on screen}
      72 : MoveVer(-4,true); {up arrow}
      75 : MoveHor(-1,true); {left arrow}
      77 : MoveHor(1,true); {right arrow}
      80 : MoveVer(4,true); {down arrow}
    end;
  until ord(ch)= 13;
  {freeze box and continue with <return> key}
end;
```

```
begin
```

```

DumpGraphData:= False;
selectscreen(1);
copyscreen; SetLineStyle(0);
    {save underlying screen and display menu box}
DefineWindow(4,11,20,35,90);
DefineWorld(4,0,0,20,20);
SelectWorld(4); SelectWindow(4);
DefineHeader(4, 'Graph Options Menu');
SetHeaderOn; SetBackground(0); DrawBorder;
DrawTextW(1,4,1, '<P> Print Graph to the printer');
    {display menu options}
DrawTextW(1,7,1, '<T> Make Title to the Graph');
DrawTextW(1,10,1, '<N> Print Table of Numbers');
DrawTextW(1,13,1, '<P> used to Generate Graph');
DrawTextW(1,17,1, '<Q> Continue to the Program');
repeat
    Option;
    case ch of      {interpret user input}
        'P': begin
            swapscreen;
                {redispays screen without menu box}
            hardcopy(false,1); {print to printer}
            copyscreen;
            ch := 'P';
        end;
        'T': begin
            leavegraphic;    {leave graphics screen}
            TitlePrompt;    {prompt for title}
            entergraphic;    {return to graphics mode}
            swapscreen;      {bring back graph}
            ShowTitle; {display title box on screen}
            copyscreen; {save graph with title box}
            ch := 'T';
        end;
        'N': begin
            DumpGraphData := True;
                {sets boolean to cause numbers }
                {to be printed}
            ch := 'N';
        end;
        'Q': ;
    end;
until ch in ['P','T','N','Q'];
if ch = 'Q' then Quit := true
else Quit := false;
end;

```

```

procedure input_Factored(Zeros_or_poles:str5;
                        NFactors:integer;var real_part,
                        imaginary_part:array3s);
{Begin processing Factored form Input-Internal routine}
var
  i,j    : integer;
  test   : real;
begin
  P[11] := '0905N01011-000101';
  P[12] := '2505N01012-000101';
  P[13] := '0907N01013-000101';
  P[14] := '2507N01014-000101';
  P[15] := '0909N01015-000101';
  P[16] := '2509N01016-000101';
  P[17] := '0911N01017-000101';
  P[18] := '2511N01018-000101';
  P[19] := '0913N01019-000101';
  P[20] := '2513N01020-000101';
  P[21] := '0915N01021-000101';
  P[22] := '2515N01022-000101';
  P[23] := '0917N01023-000101';
  P[24] := '2517N01024-000101';
  P[25] := '0919N01025-000101';
  P[26] := '2519N01026-000101';
  P[27] := '0921N01027-000101';
  P[28] := '2521N01028-000101';
  P[29] := '0923N01029-000101';
  P[30] := '2523N01030-000101';

  ClrScr; TextColor(White);    {write screen titles}
  gotoXY(1,24);
  Invvideo('Press <ESC> to change an entry');
  Center('*** Desired characteristic Polynomial
        Input***',1,1,80);
  HighVideo;
  writeln;TextColor(green);
  if Zeros or Poles = 'ZEROS' then
    writeln('NUMERATOR Transfer Function
            Input -- FACTORED Form')
  else
    writeln('Characteristic equation
            Input -- FACTORED Form');

  HighVideo;writeln;writeln;
  for j:=1 to NFactors do {type prompt strings}
    begin
      writeln('      s =                +j');
      writeln;
    end;

  str((NFactors*2+10):2,strg);
  specification := concat('N11',strg);
  Input_handler(specification, escape);
  {call the input handler}

  for j:= 1 to NFactors do
    {compute the zero values from}
    begin
      {input handler string Filvar}
      val(filvar[2*j+9 ],test,code);
      if code = 0 then Real Part[j]:=test;
      {val conversion successful if code}
    end;

```

```

        val(filvar[2*j+10],test,code);
        {=0, error otherwise}
        if code = 0 then Imaginary_Part[j]:=test;
    end;
end; {procedure Input_factored}
{Begin processing Coefficient form Input-Internal
procedure }
Procedure Input_Coeff(Zeros or Poles:Str5;
                     NCoeff:integer;
                     var Coeff: ary4);
var
    i,j,
    NCoeff_old : integer; {counters}
    test: real;           {holds old poly order if
                          {changing order}
                          {holds "val" results until validated}

begin
    P[21] := '0406N01021-000101';
    {Input-Handler descriptors for coeff}
    P[22] := '1806N01022-000101'; {form input}
    P[23] := '3206N01023-000101';
    P[24] := '4606N01024-000101';
    P[25] := '0408N01025-000101';
    P[26] := '1808N01026-000101';
    P[27] := '3208N01027-000101';
    P[28] := '4608N01028-000101';
    P[29] := '0410N01029-000101';
    P[30] := '1810N01030-000101';

    NCoeff_old:= NCoeff;

    ClrScr; TextColor(White); {print screen titles}
    gotoxy(1,24);
    invvideo('Press <ESC> to change an entry');
    gotoxy(1,1);
    writeln('*** Desired characteristic
    Polynomial input ***');
    TextColor(Green);
    if Zeros_or_Poles = 'POLES' then
        writeln('NUMERATOR Transfer Function
        Input -- COEFFICIENT Form')
    else
        begin
            write('COEFFICIENT Form ---');
            TextColor(lightmagenta);
            if not luen then
                writeln('Highest degree coefficient
                MUST be 1.0');
        end;
    end;
    HighVideo;

    if NCoeff > NCoeff_old then
        begin
            for j:=1 to NCoeff_old - NCoeff do
                for i:=NCoeff_old + 1 downto 1 do
                    Filvar[i+1] := Filvar[i];
                end;
            end;
        if NCoeff < NCoeff_old then

```

```

begin
  for j:= 1 to NCoeff old - NCoeff do
    for i:= 1 to NCoeff old + 1 do
      Filvar[i] := Filvar[i+1];
    end;
  end;

VertPos:=4;
for i:=NCoeff+1 downto 1 do
  begin
    j:=NCoeff+1 - i;
    PosCounter := (j mod 4) + 1;
    HorizPos := PosCounter * 14;
    If PosCounter = 1 then VertPos := VertPos + 2;

    if i<> 1 then{prompts for coeff input}
      begin
        msg('s +',HorizPos,VertPos);
        str(i-1:2,Exponent);
        msg(Exponent,HorizPos+1,VertPos-1);
      end;
    end;

    str((20+NCoeff+1):2,strg); {sets up and calls
                               input handler}

    specification := concat('N21',strg);
    Input_handler(specification,escape);
    for j:= NCoeff+1 downto 1 do
      begin
        val(Filvar[NCoeff+22-j],test,code);
        if code = 0 then Coeff[j]:=test;
      end;
    end;
  end;
end;

```

```

procedure characteristic_equation(var A3:ary1s;
                                var N:integer;
                                var C2:ary4);

```

{This procedure computes the characteristic equation polynomials coefficients by using the principle-minor method}

```

label 10;label 20;label 30;label 40;label 50;
label 60;label 15; label 70;label 80;label 90;
label 100;label 1;label 2;label 5;

```

```

var
  B3,A1                                :ary1s;
  integer_vector                       :ary6;
  D10 : array [1..280] of real;
  mm,ii,l,m,i,kk,nr,nc,k1,i1i,
  counter,m1m,even,nn,k,j1,n1,m1      :integer;
  temp_value,det,det_correction,
  determinant_old,determinant,value_c2,value :real;

```

```

begin
  nn:=n+1;
  for i:=1 to nn do
    begin
      C2[i]:=0.0;
    end;
  C2[nn]:=1.0;
  for m:=1 to n do
    begin
      k:=0; l:=1;
      integer_vector[1]:=1;
      goto 2;
      1:integer_vector[1]:=integer_vector[1]+1;
      2:if (l-m) > 0 then goto 90;
      if (l-m) = 0 then goto 5;
      mm:=m-1;
      for i:= 1 to mm do
        begin
          ii:=i+1;
          integer_vector[ii]:=integer_vector[i]+1;
        end;
      5:for i:=1 to m do
        begin
          for kk:=1 to m do
            begin
              nr:=integer_vector[i];
              nc:=integer_vector[kk];
              B3[i,kk]:=A3[nr,nc];
            end;
          end;
        k:=k+1;

        counter := 0;
        for i1i:=1 to M do
          begin
            for j1:=1 to M do
              begin
                A1[i1i,j1]:= B3[i1i,j1];
              end;
            end;
          for i1i:= 1 to M do
            Begin
              k1:=i1i;
              30:if A1[k1,i1i] <> 0.0 then goto 10;
              k1:=k1+1;
              if (k1-M) <= 0 then goto 30;
              goto 40;
            end;
          end;
        end;
      end;
    end;
  end;

```

```

10:if (i1i-k1) > 0 then goto 40;
if (i1i-k1) = 0 then goto 70;
for m1m:=1 to M do
Begin
  temp_value:=A1[i1i,m1m];
  A1[i1i,m1m]:=A1[k1,m1m];
  A1[k1,m1m]:=temp_value;
End;
counter:= counter+1;
70:i1:=i1i+1;
if i1 > M then goto 20;
for m1m:=i1 to M do
Begin
  if A1[m1m,i1i] = 0.0 then goto 80;
  value:=A1[m1m,i1i] / A1[i1i,i1i];
  for n1:= i1i to M do
  begin
    A1[m1m,n1]:= A1[m1m,n1] - A1[i1i,n1] * value;
  end;
80:End;
20:End;
det:=1.0;
for i1i:=1 to M do
begin
  det:=det * A1[i1i,i1i];
end;
det_correction:= exp( counter * ln(1.0));
determinant_old:=det_correction * det;
even:= counter mod 2;
if even <> 0 then goto 60;
determinant := determinant_old;
goto 50;
60:determinant:=-determinant_old;
goto 50;
40:determinant:=0.0;

50:D10[k]:=determinant;
for i:= 1 to m do
begin
  l:=m-i+1;
  if (integer_vector[l]-(n-m+1)) > 0 then goto 90;
  if (integer_vector[l]-(n-m+1)) < 0 then goto 1;
end;

m1:=n-m+1;
value_c2:=exp( m * ln(1.0));
even:= m mod 2;
for i:=1 to k do
begin
  if even = 0 then
  begin
    value_c2:=value_c2;
    goto 15;
  end;
  if even <> 0 then
  begin
    value_c2:=-1.0 * value_c2;
  end;
  15:C2[m1]:=C2[m1] + D10[i] * value_c2;
  value_c2:=exp( m * ln(1.0));
end;

end;goto 100;
90:writeln('Error in characteristic equation');
100:end;

```

```
{ This procedure uses modified BARSTOW method to
  calculate the roots of the polynomial }
```

```
PROCEDURE root_finder(var n:integer;
                      var A:ary4;
                      var u,v:ary3s;
                      var ir:integer);
```

```
{ label decleration for the GOTO statement}
```

```
label 3;label 4;label 7;label 9;label 10;label 13;
label 15;label 19;label 20;label 23;label 30;
label 32;label 33;label 34;label 36;label 49;
label 50;label 52;label 53;label 70;label 72;label 73;
label 75;label 76;label 81;label 82;label 100;
```

```
var
  irev,i,nc,m,nl,np,j,i1:integer;
  p,q,r,f,e,cbar,d,qp,pp:real;
  H,B,C: array [1..21] of real;
```

```
Begin
```

```
  irev:=ir; {take given values}
  nc:=n+1;
  for i:=1 to nc do
    H[i]:= A[i];
```

```
  p:=0.0; {initialization}
  q:=0.0;
  r:=0.0;
  3:if H[1] <> 0 then goto 4;
  nc:=nc-1;
  U[nc]:=0.0;
  V[nc]:=0.0;
  for i:= 1 to nc do
    H[i]:= H[i+1];
```

```
  goto 3;
  4:if (nc-1) = 0 then goto 100;
  if (nc-2) <> 0 then goto 7;
  r:=-H[1]/H[2];
  goto 50;
  7:if (nc-3) <> 0 then goto 9;
  p:=H[2]/H[3];
  q:=H[1]/H[3];
  goto 70;
  9:if (abs(H[nc-1]/H[nc])-abs(H[2]/H[1])) >= 0.0
    then goto 19;
  irev:=-irev;
  m:=nc div 2;
  for i:=1 to m do
    begin
      nl:=nc+1-i;
      F:=H[nl];
      H[nl]:=H[i];
      H[i]:=F;
```

```
  end;
  if Q <> 0.0 then goto 13;
  p:=0.0;
  goto 15;
  13:p:=p/q;
  q:=1.0/q;
  15:if r = 0.0 then goto 19;
  r:=1.0/r;
```

```

19:e:=5.0e-10;
B[nc]:=H[nc];
C[nc]:=H[nc];
B[nc+1]:=0.0;
C[nc+1]:=0.0;
np:=nc-1;
20:for j:= 1 to 1000 do
begin
  for i1:= 1 to np do
  begin
    i:=nc-i1;
    B[i]:=H[i] + R * B[i+1];
    C[i]:=B[i] + R * C[i+1];
  end;
  if (abs(B[1]/H[1])-e) <= 0.0 then goto 50;
  if C[2] <> 0.0 then goto 23;
  r:=r+1.0;
  goto 30;
23:r:=r - B[1]/C[2];
30:for i1:= 1 to np do
begin
  i:=nc-i1;
  B[i]:= H[i] - p * B[i+1] - q * B[i+2];
  C[i]:= B[i] - p * C[i+1] - q * C[i+2];
end;
  if H[2] <> 0.0 then goto 32;
  if (abs(B[2]/H[1])-e) > 0.0 then goto 34;
  if (abs(B[2]/H[1])-e) <= 0.0 then goto 33;
32:if (abs(B[2]/H[2])-e) > 0.0 then goto 34;
33:if (abs(B[1]/H[1])-e) <= 0.0 then goto 70;
34:cbar:= C[2] - B[2];
d:=C[3] * C[3] - cbar * C[4];
  if d <> 0.0 then goto 36;
  p:=p - 2.0;
  q:=q * (q + 1.0);
  goto 49;
36:p:= p + (B[2] * C[3] - B[1] * C[4])/d;
  q:= q + (-B[2] * cbar + B[1] * C[3])/d;
49:end;
e:=e * 10;
goto 20;
50:nc:= nc-1;
V[nc]:= 0.0;
if irev >= 0 then goto 52;

U[nc]:=1.0/r;
goto 53;
52:U[nc]:=r;
53:for i:=1 to nc do
  H[i]:= B[i+1];

goto 4;
70:nc:=nc-2;
if irev >= 0 then goto 72;
qp:=1.0/q;
pp:=p/(q * 2.0);
goto 73;
72:qp:= q;
pp:=p/2.0;

73:f:=pp * pp - qp;
if f >= 0.0 then goto 75;
U[nc+1]:= -pp;
U[nc]:= -pp;
V[nc+1]:= sqrt(-f);
V[nc]:= -V[nc+1];
goto 76;
75:if pp <> 0.0 then goto 81;

```

```
U[nc+1]:= -sqrt(f);
goto 82;
81:U[nc+1]:=-(pp/abs(pp)) * ( abs(pp) + sqrt(f));
82:V[nc+1]:=0.0;
U[nc]:= qp/U[nc+1];
V[nc]:=0.0;
76:for i:=1 to nc do
    H[i]:= B[i+2];
goto 4;
100:end;
```

LIST OF REFERENCES

1. Melsa, James L. and Jones, Stephen K., Computer Programs for Computational Assistance in the study of Linear Control Theory, McGraw- Hill, 1973.
2. Desjardins, Berthier, A User's Manual for Linear Control Programs on IBM/360, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1979.
3. Borland International, Inc., Turbo Pascal Tutor, 1986.
4. Wood Jr., Roy L., Microcomputer Based Linear, System Design Tool, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1986.
5. Borland International, Inc., Turbo Pascal Reference Manual, Version 2.0, 1983.
6. Thaler, George J., Automatic Control Systems, EC-2300 Text, Naval Postgraduate School, Monterey, California, 1986.
7. Borland International, Inc., Turbo Graphix Toolbox Reference Manual, Version 1.0, 1985.
8. Cooper, Doug and Clancy, Michael, Oh! Pascal!, W.W Norton & Company Inc., 1985.
9. Schildt, Herbert, Advanced Turbo Pascal Programming and Techniques, McGraw - Hill, 1986.

10. Miller, Alan R., Pascal Programs for Scientists and Engineers, Sybex Inc., 1981.
11. Koffman, Elliot B., Problem solving and structured Programming in Pascal, Second Edition, Addison - Wesley Publishing Company, Inc., 1985.
12. Ogata, K., Modern Control Engineering, Prentice - Hall, Inc., 1970.
13. Gilder, Jules H. and Barrus, J. Scott, Pascal Programs in Science and Engineering, Hayden Book Company, Inc., 1983.
14. Friedland, Bernard, Control System Design an Introduction to State - Space Methods, McGraw - Hill book company, 1986.
15. Problem Solver in Automatic Control Systems - Robotics, Research and education Association, 1982.
16. Thompson, Robert M., A User's Manual for Interactive Linear Control Programs on IBM/3033, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1982.
17. Vegte, John Van De, Feedback Control Systems, Prentice - Hall, Inc., 1986.
18. Melsa, James L. and Schultz, Donald G., Linear Control Systems, McGraw-Hill, Inc., 1969.
19. Luenberger, D. G., An Introduction to Observers, IEE Trans. Auto. Control, Vol. AC-16, No. 6., pp. 596-602, 1971.

20. Athans, M. and Falb, Peter L., Optimal Control, McGraw-Hill, Inc., 1966.
21. Luenberger, D. G., Observing the State of a Linear System, IEEE Transactions on Military Electronics, Vol. MIL-8, pp. 74-80, April 1964.
22. Luenberger, D. G., Observers for Multivariable systems, IEE Transactions on Automatic Control, Vol. AC-11, No.2, pp. 190-197, April 1966.
23. Joint Automatic Control Conference of the American Automatic Control Council, pp. 16-39, 1971.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3. Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
4. Professor Thaler G. J., Code 62Tr Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	15
5. Professor Cristi Roberto, Code 62Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
6. Ismail UNLU Dokuzlar koyu Kiraz - IZMIR 35894 TURKEY	2

7. Ibrahim DINCER 1
Muhabere Okulu Ogretim Kurumu
Mamak - ANKARA
TURKEY
8. Orhan BABAOGLU 1
SMC 2658
Naval Postgraduate School
Monterey, California 93943
9. Deniz Harp Okulu Komutanligi 2
Okul Kutuphanesi ve
Elektrik bolumu Kutuphanesi
Tuzla - ISTANBUL
TURKEY
10. Deniz Kuvvetleri Komutanligi 5
Kutuphanesi
Bakanliklar - ANKARA
TURKEY
11. Istanbul Teknik Universitesi 1
Elektrik Fakultesi
ISTANBUL, TURKEY
12. Orta Dogu Teknik Universitesi 1
Elektrik Fakultesi
ANKARA, TURKEY
14. Metin SELEK 1
Sogutlu Cesme,Elmali Cesme sok.
Huzur apt. Daire: 6
Kadikoy - ISTANBUL
TURKEY

13. Bogazici Universitesi 1
Elektrik Fakultesi
ISTANBUL, TURKEY
15. Sevki SEKEREFELI 1
Kiziltoprak Mah. Taskopru cad. No:22/8
ESKISEHIR / TURKEY
16. Dr. Samir I. MUSTAFA 1
Mechanical Engineering Dept.
Naval Postgraduate School
Monterey, California 93943
17. Systems Engineering Department 1
US Naval Academy
Annapolis, MD
18. NAVSEA 1
Washington D. C. 20362-5101

Thesis

U44

Unlu

c.1

Computer aided design
for linear control state
variable systems (SVS).

Thesis

U44

Unlu

c.1

Computer aided design
for linear control state
variable systems (SVS).



thesU44

Computer aided design for linear control



3 2768 000 78707 1

DUDLEY KNOX LIBRARY